NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

A STOCHASTIC SIMULATION OF A UNITED STATES NAVAL CONFLICT WITH A LAND-BASED OPPONENT: THE IMPACT OF C⁴ISR

by

Edward R. Martinez

September, 1996

Thesis Advisor: Donald P. Gaver

Approved for public release; distribution is unlimited.

19970116 114

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE September 1996.	3. REPORT TYPE AND DATES COVERED Masters Thesis

4. TITLE AND SUBTITLE TITLE OF THESIS. A STOCHASTIC SIMULATION OF A UNITED STATES NAVAL CONFLICT WITH A LAND-BASED OPPONENT: THE IMPACT OF C4ISR		5. FUNDING NUMBERS
6. AUTHOR(S) Edward R. Martinez		
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey CA 93943-5000		8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)		10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U. S. Government.

12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.	12b. DISTRIBUTION CODE

13. ABSTRACT *(maximum 200 words)*

This thesis develops a low-resolution stochastic simulation model to assess the impact of the intelligence, surveillance and reconnaissance components of C4ISR, and strike capabilities on the mission success of a United States carrier battle group (CVBG). The simulation uses a stochastic approach to model a two-day conflict between a CVBG and a land-based enemy which incorporates the randomness and uncertainty inherent in warfare. The simulation is implemented as a C++ computer program to develop a tool to analytically exercise a prospective new system in order to predict its possible effect on combat operations. Experiments were run which simulated a two-day battle in which the United States CVBG sensor availability, sensor accuracy, and weapons availability were varied to study their affect on the outcome of the battle. Statistical analysis techniques are used to quantitatively measure the results of the battle as the sensor and weapon parameters change.

14. SUBJECT TERMS information warfare, carrier battle group operations, weapons, simulation, stochastic model			15. NUMBER OF PAGES 148
			16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL

A STOCHASTIC SIMULATION OF A UNITED STATES NAVAL CONFLICT WITH A LAND-BASED OPPONENT: THE IMPACT OF C⁴ISR

Edward R. Martinez
Lieutenant, United States Navy
B.S., United States Naval Academy, 1989

Submitted in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN OPERATIONS RESEARCH

from the

NAVAL POSTGRADUATE SCHOOL
September 1996

Author: _____

Edward R. Martinez

Approved by: _____

Donald P. Gaver, Thesis Advisor

_____

Patricia A. Jacobs, Second Reader

_____

Frank C. Petho, Chairman
Department of Operations Research

ABSTRACT

This thesis develops a low-resolution stochastic simulation model to assess the impact of the intelligence, surveillance and reconnaissance components of C^4ISR, and strike capabilities on the mission success of a United States carrier battle group (CVBG). The simulation uses a stochastic approach to model a two-day conflict between a CVBG and a land-based enemy which incorporates the randomness and uncertainty inherent in warfare. The simulation is implemented as a *C++* computer program to develop a tool to analytically exercise a prospective new system in order to predict its possible effect on combat operations. Experiments were run which simulated a two-day battle in which the United States CVBG sensor availability, sensor accuracy, and weapons availability were varied to study their affect on the outcome of the battle. Statistical analysis techniques are used to quantitatively measure the results of the battle as the sensor and weapon parameters change.

DISCLAIMER

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the United States Government.

The reader is cautioned that computer programs developed in this research may not have been exercised for all cases of interest. While every effort has been made, within the time available, to ensure that the programs are free of computational and logic errors, they cannot be considered validated. Any application of these programs without additional verification is at the risk of the user.

TABLE OF CONTENTS

EXECUTIVE SUMMARY

A. PURPOSE

The purpose of this thesis is to develop a low-resolution stochastic simulation approach to model a naval theater conflict against a land-based enemy which incorporates the aspects of the randomness and uncertainty inherent in warfare. The simulation is designed to assess the impact of intelligence, reconnaissance, and surveillance information, derived from internal and external sources, and strike capabilities (weapons types, numbers, and characteristics), specifically and illustratively on the mission success of a United States carrier battle group (CVBG) that is tasked to provide support for a multinational force operating in a hostile region.

B. BACKGROUND

Command, control, communications, computer systems, intelligence, surveillance, and reconnaissance (C^4ISR) and superior strike capability are two keys to the future success of United States carrier battle group operations. A CVBG is frequently tasked to exert the policy of the United States in remote areas of the world against powerful military nations. The leaders that make the decisions on the type of C^4ISR and weapons assets to develop and procure need tools to analytically exercise a prospective new system in order to predict its possible effect on combat operations.

The proposed modeling approach examines a realistic naval engagement in which a group of multinational force ships is initially in the Persian Gulf and wishes to transit out of the

area through the Straits of Hormuz. The transit is opposed by Iranian land-based surface-to-surface missile launchers and air forces. The United States government dispatches a CVBG to protect the multinational force as it transits the straits by destroying Iranian mobile surface-to-surface missile launchers and defeating Iranian fighter, attack aircraft, and reconnaissance aircraft.

C. RESULTS

A stochastic simulation model to assess the impact of information and strike capabilities for this situation has been programmed in the C++ computer language. Initial force structures, sensor, and weapons capabilities were established using general unclassified military publications. Computer experiments were then performed which simulated a two-day conflict between the United States naval carrier battle group and Iran. The initial conditions of the computer runs were varied to affect the sensor and weapons availability, and sensor accuracy for the United States and multinational forces. Statistical analysis techniques were used to examine the results of the computer simulation runs to determine how the varied sensor and weapon capabilities affected the outcome of the battle.

The analysis of the simulation results show that the use of an additional aircraft carrier combined with the availability of satellite intelligence enables the U. S. carrier battle group to complete its mission. On the basis of this initial computer experiment, the proposed simulation modeling approach shows promise of having informative predictive capabilities, but more detail in the models is required.

I. INTRODUCTION

A. BACKGROUND

Command, control, communications, computer systems, intelligence, surveillance, and reconnaissance (C^4ISR) is presently receiving a significant amount of attention in the Department of Defense. The United States (U. S.) armed services are concerned with acquiring nearly perfect knowledge of enemy units, movements, and plans while denying the enemy the same information. C^4ISR includes the assignment of the sensors used to obtain data, communication links used to transmit and receive data, the capabilities and priorities of the machines and men that process raw data, and the commander's perception of the military situation on the basis of the analyzed end product. The leaders that make decisions on the type of C^4ISR assets to develop and procure need tools to analytically exercise a prospective new system in order to predict its possible effect on combat operations. Modeling and simulation tools are, and have potential to be used extensively by the Department of Defense to influence these decisions. This thesis used a specific naval conflict to construct a stochastic simulation model that incorporates some of the randomness and uncertainty inherent in warfare. The model is programmed as a computer simulation in the $C++$ computer language. Simulation trials are run to explore the effects of changing conditions that relate to C^4ISR which are programmed into the model on the outcome of the naval conflict.

The scenario that the model was designed for is a realistic naval scenario that may be of interest to U. S. military planners. The scenario envisions a group of multinational force units that is in the Persian Gulf and that wishes to transit out of the area through the Straits of Hormuz; its transit is opposed by Iranian land-based surface-to-surface missile launchers and air forces. The multinational unit's exit is assisted by a U. S. carrier battle group (CVBG). The interplay of C^4ISR and strike capabilities is judged by use of the exploratory simulation model that is implemented in this thesis.

B. THESIS OBJECTIVE

The objective of this thesis is to develop a stochastic modeling approach that can assess the impact of intelligence, reconnaissance, and surveillance information, derived from internal and external sources, and strike

capabilities (weapons types, numbers, and characteristics), on the mission success of a U. S. CVBG in the scenario outlined above. Simulation is used to study the outcomes of the stochastic model.

C. APPROACH

This thesis develops a low-resolution closed-loop stochastic simulation model of a specific naval scenario. The element of chance is incorporated into the model by assigning specific probabilities for the occurrence of certain significant events (e.g. detection of a target, weapon hits on a target, operation of radar systems, etc.). When the model is tasked to determine the outcome of a significant event, the outcome is simulated using appropriate probabilities. The model design attempts to represent the impact of various types of informational sensors (own unit and external), the battlefield perception drawn from the combined sensor information, and the subsequent number of assets utilized and losses experienced while attempting to complete an assigned mission. This modeling approach has been realized as a time-step simulation programmed in the $C++$ computer language to produce quantitative results. The units that comprise the Iranian and United States forces, and the sensor and weapons capabilities of each force were defined for this model to allow for quantitative analysis of simulation runs. Analysis of the simulation results using a specific initial force structure and unit characteristics has shown that this model does not contain sufficient detail to properly assess the affect of C^4ISR. This model presents an approach that has promise of appraising the affect of C4ISR on U. S. naval operations if more detail is programmed into the computer simulation.

II. SITUATION

A. GENERAL DESCRIPTION

The situation considered is that a multinational force, consisting of a few U. S. combatants escorting

merchant ships, is located in the Persian Gulf. Iran decides to prevent the force from moving through the Straits

of Hormuz, and threatens aircraft and surface-to-surface missile (SSM) attacks on the vessels. The prospect of a

complex war in this region precludes the U. S. from obtaining any friendly air or littoral support. The U. S.

sends a CVBG to protect the force whenever it chooses to transit through the straits to the Arabian Sea. The

situation is shown in Figure 1.
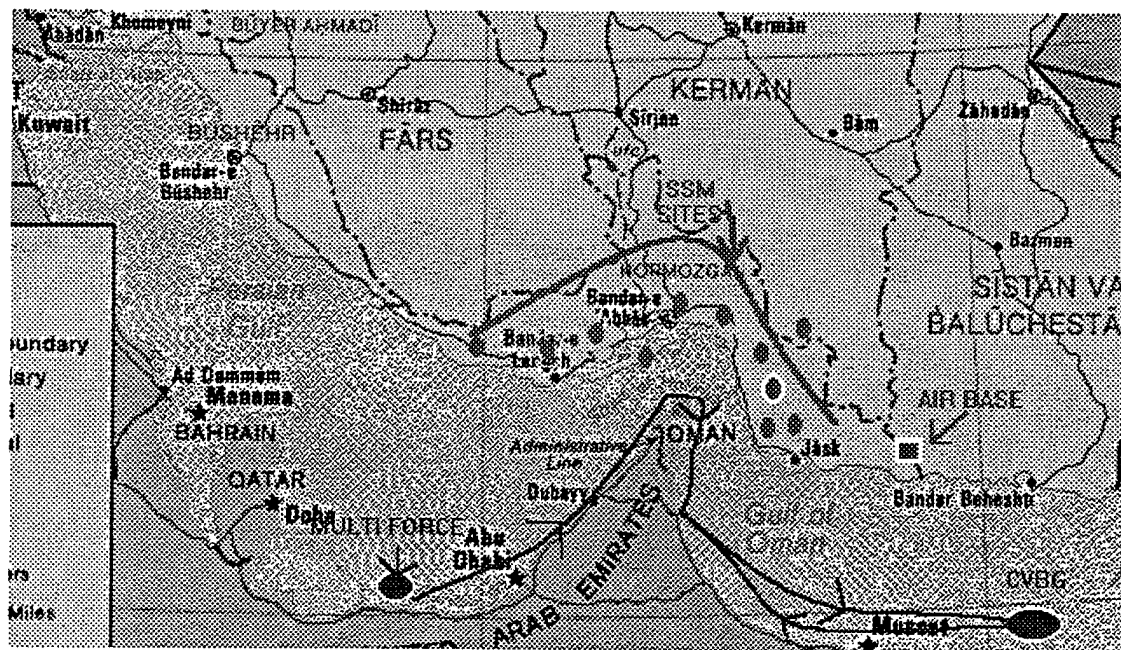
Figure 1. Conflict Region and Approximate Locations of Opposing Forces

The goal of the multinational force is to transit the straits into the Arabian Sea, but to do so while

experiencing minimal damage. The CVBG is tasked to protect the multinational force by destroying Iranian

SSM weapon sites around the Straits while minimizing losses of aircraft and damage to the battle group. The

goal of the Iranian forces is to prevent both the multinational force from completing its transit, and to maximize

damage to the U. S. naval forces assigned to protect the multinational force.

3

B. ORDER OF BATTLE

The multinational force consists of 1 DDG (Burke class with vertical launch system (VLS)), 1 DDG (Spruance class with VLS), and 2 Merchants. The U.S. CVBG consists of 1 CVN (Nimitz) and an augmented aircraft contingent which includes 46 F/A-18 strike aircraft, 30 F-14 fighter, 4 E-2C early warning aircraft, and 4 reconnaissance aircraft, 2 CGs (Ticonderoga class w/VLS), 3 DDGs (Spruance/Burke w/ VLS), and 1 AOR. The mission of the U.S. carrier is to deter Iranian air strikes by destroying their attack aircraft, attacking their air bases, destroying their SSM launchers, and eliminating their primary sensors (reconnaissance aircraft and long range radar sites); the warships are tasked to provide defense for the aircraft carrier and strike land-based targets using Tomahawk cruise missiles. The United States force can also have two satellites available for overhead visual and electronic intelligence gathering.

The Iranian force consists of land-based attack, fighter, and reconnaissance aircraft squadrons, SSM, and SAM sites. The Iranian's primary sensors are land-based long range radar, and reconnaissance aircraft.

C. STUDIED SITUATION

The assumptions of the model for the scenario are as follows. In the computer implementation, an external file is used to define the initial force structure for both sides. This external database defines each unit by type, weapons systems (type, capacity, range), and movement patterns. The characteristics, tactics, and locations of all units and weapons incorporated in this model are derived from unclassified general publications and information. The initial database is designed to force a two-day battle between United States and Iranian forces.

As the battle commences, the multinational force is located in the southern Persian Gulf, transiting towards the Straits of Hormuz, and the CVBG is transiting towards a rendezvous point southeast of the straits. The tracks that all ships will follow during the two-day battle are pre-determined, and only their rate of movement varies around an average transit speed. The multinational force travels through the Straits of Hormuz, joins the CVBG, and transits towards the Arabian Sea. Both the CVBG and the multinational force follow tracks that maintain them at the maximum range from Iran while remaining in sufficiently deep water. The Iranian forces commence hostilities as soon as they detect any naval units, and the U. S. forces launch pre-emptive strikes when they detect the Iranians. The battle concludes after 48 hours, and on the average the CVBG and the multinational force (assuming the multinational force survived) have reached the entrance to the Arabian Sea.

4

When the United States has satellite assets available, the satellites examine regions of the Iranian countryside where it is suspected that operating air bases and mobile SSM launchers are located. These pre-determined flight patterns result in possible detections. The satellite capability is represented by two sensors. One of the sensors performs a three hour scan of a region of Iran where threat contacts are located and then takes nine hours to orbit the earth before returning to conduct another scan of the same area. The other sensor performs a 2.5 hour scan over a different area where threat contacts are located and then takes six hours to orbit the earth before returning to conduct another scan of the same area. All other sensors are located on moving or stationary objects and the area where one of these sensors has the possibility of detecting an enemy unit depends on both the location and range of a given sensor, which is defined for each sensor in the initial data base.

D. TACTICAL ASSUMPTIONS

1. This scenario excludes any combat between naval units, other than between carrier aircraft and land-based strike and reconnaissance aircraft.

2. Intelligence sources are restricted to the sensors in the simulation, and Human Intelligence (HUMINT) is not built into the model.

3 The naval units attached to the CVBG or multinational task force move in formation. There is no independent steaming of ships, and they proceed along pre-determined courses. The movement track of the CVBG ensures that all units remain out of range of Iranian SSM launchers. The CVBG units therefore are only susceptible to attacks by aircraft.

4. The naval units are considered to be independent entities for detection purposes. Therefore, the detection of one unit does not reveal the location of another unit. In practice, identifying the location of one ship of a naval task force implies that other ships are nearby, but this is ignored in this model.

5. The dissemination of information between friendly units is assumed to be perfect and effectively instantaneous. This implies that information available to one unit is available with only a small negligible time delay to all units, so a common intelligence picture can be formulated.

6. The Iranian forces utilize continuous radar coverage to detect an incoming U.S. strike. The United States Navy operates using the Aegis radar system, which is resident on Ticonderoga class cruisers, as the primary air defense asset protecting a carrier battle group. Use of the Aegis radar system exposes the location of

5

the ship it is located on to the enemy electronic intelligence sensors. The U. S. believes that the benefits of using this system outweigh exposing the location of naval assets to enemy electronic intelligence sensors (ELINT).

7. SAM sites are modeled as if they are collocated to an air base, SSM, or command and control center and are represented as a close-in weapons system (defensive asset). They are assumed to only engage an enemy unit that is attacking the facility that is collocated with the SAM site.

8. Once a missile launch occurs, it may only be defeated by deception, jamming, or a close-in weapons system. Aircraft are not given the capability to shoot missiles out of the sky.

9. Iranian supply depots that reload mobile SSM launchers with missiles are distributed throughout the model of the Iranian country. They cannot be detected by sensor assets (i.e. are effectively undetectable), and therefore are never targeted or destroyed. This assumption is not necessarily realistic, and can be changed in future revisions.
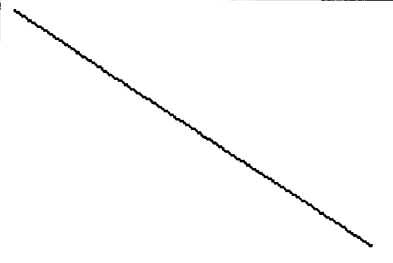
10. The unit that has the least distance between itself and an enemy target, and that has a weapon onboard that can attack the enemy, will fire at the target.

Note: Various of the restrictive assumptions made above can be altered in later revisions of the software.

III. MODEL

A. GEOGRAPHIC REPRESENTATION

The region is represented by a 700 X 400 mile map consisting of 20 X 20 mile grids (700 grids, 20 rows by 35 columns). The 20 mile grids maintain a ship within the same grid for at least 30 minutes during a transit (the typical transit speed of a merchant is 15 knots, and a naval warship is 20 knots). Each grid is numbered sequentially as shown in Figure 2. A grid contains information identifying the grid boundaries, the type of grid (land or water), the general weather condition over the grid, separate lists of United States and Iranian units that are located on the grid, and separate lists of the Iranian units that the U. S. forces have detected within the grid, and U. S. units that the Iranian forces have detected.

1	2	....	34	35
36	37	....	69	70
. . . .	. . . .		. . . .	. . . .
666	667	....	699	700

Figure 2. Geographic grid and numbering system

The database to build this region is located in Appendix A and is displayed graphically in Figure 3. Each list of units that are located on the grid identify individual units by a unit number and Cartesian location, i.e. they specify the current ground truth. Both the U. S. and Iranian perception of the location of enemy units, which is realistically imperfect, is maintained using two additional lists that reflect the perceived (not actual) inhabitants of this grid system. Movement of naval units is restricted to sea grids, and ground units to ground grids, while aircraft can move freely among both types of grids.

7
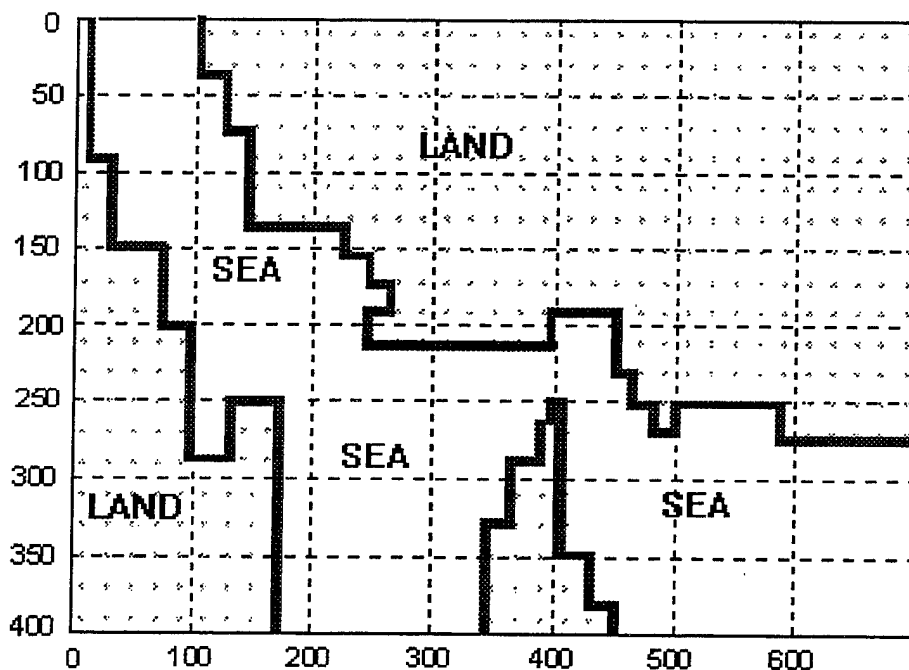
Figure 3. Geographic Data Base as Stored in the Computer Simulation

B. COMBAT UNITS

1. Overview

Status of the military units is maintained in a 40-element array. Each individual platform (U. S. ship, U. S. reconnaissance aircraft, U. S. satellite, Iranian air base, Iranian SSM launcher, Iranian reconnaissance aircraft, or Iranian radar station) is identified by a unique unit number and the information describing the operational parameters of each unit; individual parameters are modified whenever an event occurs that changes a parameter during the course of the simulation. The information that is stored for each unit is as follows:

a. Unit Characteristics

1) unit number
2) type of unit
3) current electronic emission status
4) number of hits to kill
5) number of hits against unit
6) unit is targeted (incoming weapon)

b. Location/Movement

1) grid location
2) x - coordinate
3) y - coordinate
4) motion status

8

5) next waypoint
6) time at current waypoint
7) course
8) speed
9) nominal operating speed
10) waypoints (six different waypoints are stored for each unit)
11) loiter time at a waypoint (six different loiter times are stored for each unit)

Note: a waypoint is a position defined by X and Y coordinates, that a unit is to pass through at some time during the conduct of the simulation. They are sequentially numbered from one to six, with the number indicating the order in which a unit will pass through each waypoint.

c. Weapons Capability

1) number of missiles or attack aircraft carried
2) effective strike range of missile or attack aircraft
3) number of fighter aircraft
4) effective range of fighter aircraft

Note: effective range of an aircraft is the number of miles the aircraft can travel from its home base (assumes normal operation speed) before it must commence its return journey (i.e. sufficient fuel).

d. Sensor Capability

1) surface sensor maximum detection
2) air sensor maximum detection range
3) ESM sensor maximum detection range

The database, which specifies the parameters of the characteristics of each unit, constructed for use in the performance of computer trials and analysis in this thesis is included in Appendix B. The base operational parameters of each unit (speed, weapons load out, weapons range, number of aircraft, sensor ranges) are estimated from unclassified publications which describe the military assets of the United States and Iranian governments; (Sharpe, 1996) and (Gunston, 1980). Specific initial parameters are changed during testing and exercise of the simulation for analysis. The total numbers of weapons fired during the conflict, and the subsequent losses, are stored for analysis purposes as separate variables.

2. Weapons

The U. S. weapon systems that are modeled are ship-launched land-attack cruise missiles (Tomahawk conventional missiles), carrier-based fighter aircraft (F-14) and carrier-based attack aircraft (F/A-18). The Iranian weapon systems modeled are land-based fighter (F-14) and attack aircraft, and surface-to-surface missiles (Exocet) launched from mobile land-based platforms. Each unit has a limited supply of weapons that is carried onboard. Aircraft carriers and air bases are given an initial number of

fighter and attack aircraft as its weapons, and U. S. naval warships carry Tomahawk missiles. The Iranian SSM launchers are the only units that can replenish their weapons supply during the simulation. Each SSM launcher begins the simulation with an initial loadout of M missiles, and, when they are expended, the SSM launcher transits to a supply depot and receives new missiles. Other types of weapons that are not represented in an effort to simplify the model could be easily added later as an improvement of the model.

Each unit maintains counters that keep track of the number of weapons it has available at a given time, and variables which state the effective range of each weapon carried. After a unit assigns a weapon to engage a target, the number of weapons available for the unit is correspondingly decreased. The model also maintains counters that track the total numbers of each weapon type fired by the U. S. and Iranian forces, and the number of weapons destroyed by the defensive mechanisms of their intended targets.

3. Sensors

The different types of intelligence gathering systems that are represented in this model include surface search and air search radar (ship and land based), airborne radar, visual detection by pilots, electronic support measures (ESM), satellite imagery, and satellite electronic intelligence (ELINT). The sensors are modeled using a cookie-cutter approach. In the model each unit is given three different types of sensors with their associated effective detection ranges, and probabilities of detecting a contact given that it is within the sensor's effective detection range. The three sensors are designated surface search, air search, and ESM search. For instance, the surface search sensor represents a surface search radar for the carrier, while it represents a pilot's eyes for a reconnaissance aircraft. All units with the exception of the mobile SSM launchers and reconnaissance aircraft are assumed to be constantly emitting (operating radar). On every hour and half-hour the SSM launcher decides whether it operates its radar for the next half-hour. A Bernoulli trial (probability equal to 0.5) determines whether the launcher operates its radar continuously during the next half-hour. If the SSM launhcer is moving, this trial is not conducted and its radar remains off. Using this approach it is possible that the SSM launcher continuously operates its radar for long periods of time. A more complex model should be used to represent the SSM launcher radar operations which takes into account the location of enemy surface units and reconnaissance aircraft which can detect the SSM launcher. When the SSM launchers are transiting, or are at

10

the missile reload site, they do not operate their radar. The U. S. and Iranian reconnaissance aircraft operate

their radar whenever they are in flight, and secure their radar the moment they land.

C. DECISION MODULES

1. Movement

a. United States Naval Units

Each ship has a set of W ($W = 6$ for the simulation runs conducted in this thesis) waypoints

designated prior to running the simulation. The ships will transit in straight line paths between waypoints at

speed V_{ship}, where V_{ship} is distributed as a normal random variable with mean μ_{Vship} (nominal operating speed)

and variance σ_{Vship}^2 conditioned to be positive. To simulate this random variable, the simulation re-computes the

value of V_{ship} until a value greater than zero is obtained. The variance accounts for deviations in course and

speed attributed to variations in wind and current conditions, personnel errors, and equipment operational

variability. The equations for computing the course and sampling for the speed of the ship are shown in

equations (1) and (2). All courses in this simulation are computed and stored using the radian as the unit of

measurement.

$$
\text{Course} = \begin{cases} \pi/2 & \Delta x = 0, \Delta y > 0 \\ -\pi/2 & \Delta x = 0, \Delta y < 0 \\ \arctan(\Delta x/\Delta y) & \Delta x \neq 0 \end{cases} \tag{1}
$$

$$
V_{ship} \sim N^+(\mu_{vship}, \sigma_{vship}) \tag{2}
$$

where $N^+(\mu_{vship}, \sigma_{vship})$ represents the distribution of a normal random variable with mean μ_{vship}
and variance σ^2_{vship} that is conditioned to be positive

Once a ship reaches a waypoint, it may loiter at the waypoint throughout a pre-determined time interval or it

may immediately commence its transit to the next waypoint. If the ship is programmed to remain at a waypoint,

its motion will be stopped until it reaches the time for it to commence the transit to the next waypoint. The

computed track of each ship will not allow it to traverse land grids and maintains each ship within the

geographic boundaries. The algorithm that determines ship motion is described in Appendix C.

Improvements to this model should allow both the carrier battle group and multinational force ships

to react to incoming strike aircraft, missiles, and reconnaissance aircraft by changing their course and speed to

avoid them. The enemy perception module that is described later does not compute an enemy target track. In

11

the current model the motion of the multinational force commences at the beginning of the simulation; an improved model could assess the number of Iranian assets destroyed by the CVBG strikes, and start the multinational force movement through the straits after a set amount of Iranian weapons capability has been destroyed.

b. Iranian Mobile Surface-to-Surface Missile Units (SSM)

Mobile SSM sites will move from their present location to a randomly determined position (waypoint), or to the missile reload site. On the hour, and half-hour, the SSM unit will use a Bernoulli trial (probability defined in the initial data base) to decide whether it should maintain its position, or compute a new position. New positions are computed using equations (3) and (4). The new X position ($Xssm_{new}$) is computed by drawing an independent random variable from a normal distribution with mean zero and standard deviation σ_{X_change} and adding this number to the current X position. The new Y position ($Yssm_{new}$) is computed similarly using a normal distribution with mean zero and standard deviation σ_{Y_change}. The σ_{X_change} and σ_{Y_change} values represent a typical movement distance for the SSM launcher (one standard deviation) and is defined by the user of the model, the value of 10 miles was arbitrarily selected for quantitative analysis. The course to travel to this new position is computed using equation (1).

$$Xssm_{new} = N(0, \sigma_{X\_change}) + X_{current\_position} \qquad (\sigma_{X\_change} = 10) \qquad (3)$$
$$Yssm_{new} = N(0, \sigma_{Y\_change}) + Y_{current\_position} \qquad (\sigma_{Y\_change} = 10) \qquad (4)$$

where $N(0, \sigma_{X\_change})$ represents the distribution of a normal random variable with mean 0 and variance σ_{X_change}, and $N(0, \sigma_{Y\_change})$ represents the distribution of a normal random variable with mean 0 and variance σ_{Y_change}.

The new position may be toward the coastline (closer to enemy naval ships) or inland, and the direction is purely by chance. On the average the mobile SSM launchers will remain in the same general area, which maintains the launchers in range of ships transiting near the Iranian coast, but confounds the United States intelligence picture because of random changes in location.

Once a new position is computed, and before the SSM starts moving, two checks are performed to ensure the new position is acceptable. The first check ensures the new position is within the boundaries of the geographic map. If the position is off the map, equations (5) and (6) compute a new position that is within the geographic boundaries. This check first determines if either the new X or Y coordinate ($Xssm_{new}, Yssm_{new}$) is

12

negative; if a coordinate is negative it is set to zero (Xleft_boundary = Ylower_boundary = 0), which is an acceptable X or Y coordinate. It then determines if the new X coordinate is greater than the maximum X coordinate of the map (Xright_boundary = 700); if it is greater, it is set equal to this maximum value. If the new Y coordinate is greater than the maximum Y coordinate (Yupper_boundary = 400), it is similarly set equal to its maximum possible value. This boundary check is applied for any unit that has just computed a new position to transit toward.

$$
Xssm_{new} = \begin{cases} X_{left\_boundary} & \text{if } Xssm_{new} < X_{left\_boundary} \\ X_{right\_boundary} & \text{if } Xssm_{new} > X_{right\_boundary} \\ Xssm_{new} & \text{if } X_{left\_boundary} < Xssm_{new} < X_{right\_boundary} \end{cases} \quad (5)
$$

$$
Yssm_{new} = \begin{cases} Y_{lower\_boundary} & \text{if } Yssm_{new} < Y_{lower\_boundary} \\ Y_{upper\_boundary} & \text{if } Yssm_{new} > Y_{upper\_boundary} \\ Yssm_{new} & \text{if } Y_{lower\_boundary} < Yssm_{new} < Y_{upper\_boundary} \end{cases} \quad (6)
$$

The second check that is performed ensures the new position is located on a numbered grid that is designated land. The first step of this check determines the numbered grid ($Grid_{new}$) where the unit is to be located once its complete its journey to its new position ($Xssm_{new}, Yssmn_{ew}$). If $Grid_{new}$ is a land grid, this new position is acceptable and the check is complete. If the $Grid_{new}$ is a sea grid, the position is changed using equations (7) and (8) which compute a new position that has a better chance of being within a land grid. In equation (7) and (8) $Grid_{present}$ is the grid where the unit is presently located. $X_{correction}$ and $Y_{correction}$ are numbers used to shift $Xssm_{new}$ and $Yssm_{new}$ so that the new SSM position is located on a numbered grid which is a land grid. The values of $X_{correction}$ and $Y_{correction}$ are set equal to one quarter of the length and width of a grid. For this simulation, the grids are 20 miles by 20 miles and the values of the $X_{correction}$ and $Y_{correction}$ are both five miles. This check is repeatedly performed until the $Grid_{new}$ is a land grid.

$$
Xssm_{new} = \begin{cases} Xssm_{new} + X_{correction} & \text{if } Grid_{new}(col) - Grid_{present}(col) < 0 \quad (7) \\ Xssm_{new} - X_{correction} & \text{if } Grid_{new}(col) - Grid_{present}(col) > 0 \end{cases}
$$

$$
Yssm_{new} = \begin{cases} Yssm_{new} + Y_{correction} & \text{if } Grid_{new}(row) - Grid_{present}(col) < 0 \quad (8) \\ Yssm_{new} - Y_{correction} & \text{if } Grid_{new}(row) - Grid_{present}(col) > 0 \end{cases}
$$

Once an acceptable new position (waypoint) is identified, the SSM launcher commences its journey after a one minute time delay at speed V_{ssm}, where V_{ssm} is distributed as an independent normal random variable with mean μ_{Vssm} (nominal operating speed) and variance σ_{Vssm}^2 conditioned to be positive. V_{ssm} is computed using equation (9). The simulation re-computes the value of V_{ssm} until a value greater than zero is obtained to produce a replication of the conditioned normal random variable.

13

$$V_{ssm} \sim N^+(\mu_{vssm}, \sigma_{vssm}) \qquad\qquad (9)$$

If the SSM is moving on the hour or half-hour, it will not stop its journey to compute a new random position, but will continue on its present course and speed until it reaches the waypoint. If the SSM has fired all of its missiles, it will commence a transit to its assigned missile depot to replenish its missile battery. The course and speed to reach the depot are calculated using equations (1) and (9). Once the SSM has reached the missile depot, it will loiter for R minutes ($R = 60$), during which time period the missile battery will be loaded with its full complement of missiles. Once the reload process is complete, a new waypoint will be chosen using equations (3) and (4). The original intent was for the SSM launcher to return near its initial position and then commence its normal movement pattern. The computer simulation maintains the launcher near the reload site for the remainder of the simulation. This is an incorrect implementation of the simulation and should be corrected in future revisions. The algorithm that determines SSM movement is described in Appendix C.

c. United States and Iranian Reconnaissance Aircraft

The reconnaissance aircraft fly in straight line paths between three waypoints at speed V_{air}, where V_{air} is a normal random variable with mean μ_{Vair} (nominal operating speed), and variance σ_{Vair}^2 conditioned to be positive. The random speed accounts for deviations in course and speed resulting from operational and equipment variations. An independent replication of V_{air} is drawn each time the reconnaissance aircraft departs a waypoint. Every reconnaissance aircraft is assigned a unique search pattern in the data base which is computed with respect to the location of the air base or current location of the aircraft carrier on which it is located. Every unit has a position which is defined by an X and Y coordinate. A new position can be defined in terms of the direction and distance from a current X coordinate and the direction and distance from a current Y coordinate. The following procedure defines the east (Xdirection = 1) or west (Xdirection = -1) direction a new X coordinate is located with respect to the current X coordinate and stores it in the variable $X_{direction}$. A new X coordinate is then computed by multiplying the $X_{direction}$ by a specific X distance, denoted $X_{travel\_distance}$, and adding this product to the current X coordinate. A similar procedure is then used to compute the Y coordinate using the variables Ydirection (north = -1, south = 1) and Ytravel_distance. The travel distances are the distances in miles

14

that an aircraft will travel in the x and y directions to reach its next waypoint. These distances are constant and set prior to the commencement of a simulation run in the initial data base

A reconnaissance aircraft flies from its aircraft carrier or air base to its first waypoint (X[1], Y[1]) which is computed using the carrier or air base as the current position and the variables $X_{direction}$, $Y_{direction}$, $X[1]_{travel\_distance}$, and $Y[1]_{travel\_distance}$. In the case of an Iranian reconnaissance aircraft, all variables are defined in the simulation or the initial data base and do not change during the simulation. For U. S. reconnaissance aircraft, the only variable that is not permanently defined is the $X_{direction}$, which is computed to be the same east or west direction as the carrier's present course. After the first waypoint is computed, the second waypoint (X[2], Y[2]) is computed with the first waypoint(X[1],Y[1]) defined as the current position, and the variables $X_{direction}$, $Y_{direction}$, $X[2]_{travel\_distance}$, and $Y[2]_{travel\_distance}$. The same $X_{direction}$ and $Y_{direction}$ variables are used for this computation, but for an Iranian aircraft, the product of the $X_{direction}$ and $X[2]_{travel\_distance}$ is subtracted from the current X coordinate instead of added as done in computing the first waypoint. The third waypoint is not defined until the aircraft has arrived at the second waypoint. The third waypoint(X[3],Y[3]) is always set equal to the X and Y coordinates of the air base or aircraft carrier from which the aircraft started its flight. Since an aircraft carrier is constantly changing position, a U. S reconnaissance aircraft is required to compute a new third waypoint (X[3],Y[3]) every minute until it reaches the location of the carrier and completes its flight. Figure 4 shows an example Iranian reconnaissance aircraft flight pattern, which is indicated by the solid lines and

15

determined using equations (10) through (15).
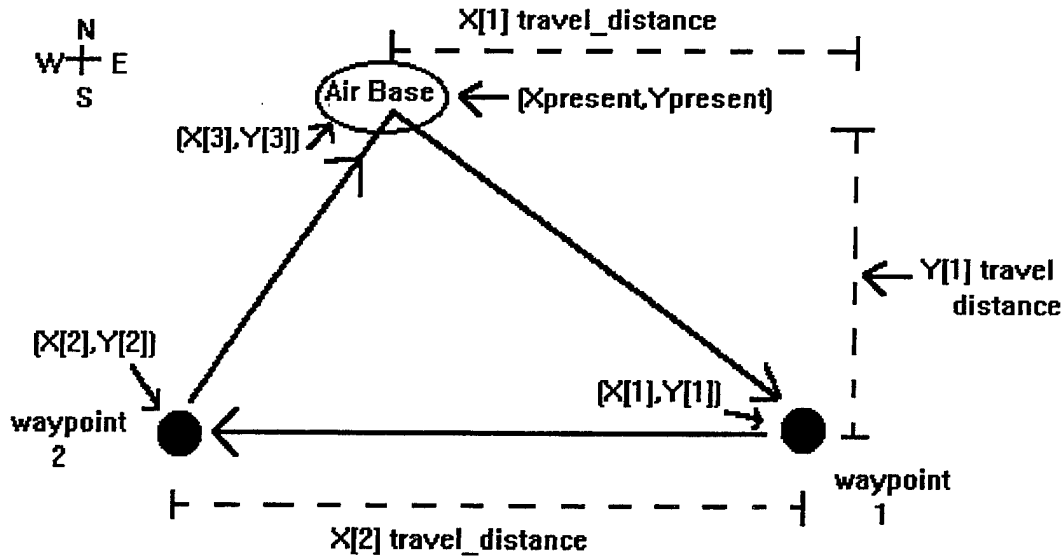
Figure 4. Iranian Reconnaissance Aircraft Search Pattern

$X_{direction}$ =   1 (fly east) or -1 (fly west) {set from initial data base})
$Y_{direction}$ =   1 (fly south)

$Y[1] = Y_{present\_location} + Y_{direction} \bullet Y[1]_{travel\_distance}$ (10)
$X[1] = X_{present\_location} + X_{direction} \bullet X[1]_{travel\_distance}$ (11)
$Y[2] = Y[1]$ (12)
$X[2] = X[1] - X_{direction} \bullet X[2]_{travel\_distance}$ (13)
$X[3] = Air Base_{xpresent\_location}$ (14)
$Y[3] = Air Base_{ypresent\_location}$ (15)

$V_{air} \sim N^+(\mu_{vair}, \sigma_{vair})$ (16)

The Iranian aircraft departs from the location of the air base and travels in a straight line to its first

waypoint(X[1],Y[1]). The aircraft then computes a new course and speed to reach the second waypoint

(X[2],Y[2]) and commences travel.. The aircraft then flies directly back to the air base(X[3],Y[3]). The course

and speed for each leg of an Iranian reconnaissance aircraft flight pattern are computed using equations (1) and

(16).

The two waypoints a U. S. reconnaissance aircraft flies through are computed prior to launch, and

depend on the present location of the aircraft carrier. Figure 5 provides an example of a U. S. reconnaissance

16

aircraft search pattern, which is indicated by the solid lines and determined using equations (10) through (12), and (17) through (21). The definition of the $X_{direction}$ variable determined in Equation (17) causes a U. S. reconnaissance aircraft to fly a pattern away from the carrier, but in the same direction of the carrier's motion east or west. This allows the aircraft to detect threats that are located in the area of the carrier's intended movement.

A U. S. aircraft departs from the current location of the carrier and travels in a straight line to its first waypoint (X[1],Y[1]). The aircraft then determines a new course and speed and commences travel to the second waypoint(X[2],Y[2]). The aircraft then returns to the aircraft carrier(X[3],Y[3]). During its return flight to the carrier, the aircraft computes a new course and speed every minute to account for the movement of the aircraft carrier. The course and speed for each leg of a U. S. aircraft flight are computed using equations (1) and (16).
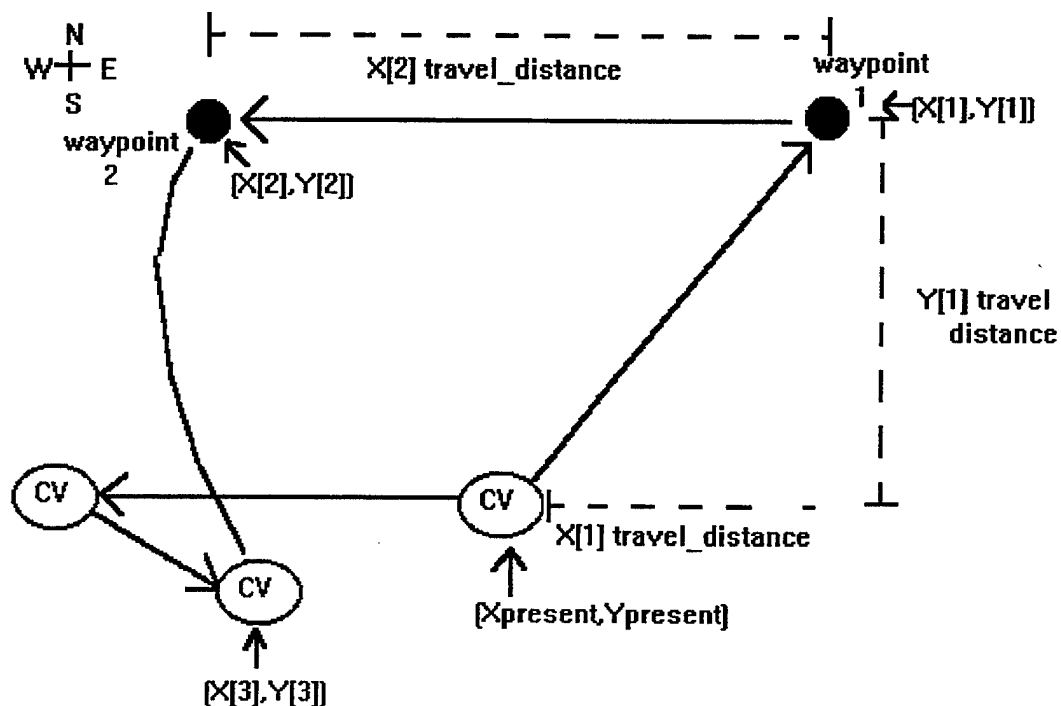
Figure 5. United States Reconnaissance Aircraft Search Pattern

$$X_{direction} = \quad 1 \text{ (fly east)} \qquad 0 < CV_{course} < \pi/2 \qquad (17)$$
$$\qquad\qquad -1 \text{ (fly west)} \qquad \pi/2 < CV_{course} < \pi \qquad (18)$$
$$Y_{direction} = \quad -1 \text{ (fly north)}$$

$$Y[1] = Y_{present\_location} + Y_{direction} \bullet Y[1]_{travel\_distance} \qquad (10)$$
$$X[1] = X_{present\_location} + X_{direction} \bullet X[1]_{travel\_distance} \qquad (11)$$
$$Y[2] = Y[1] \qquad (12)$$
$$X[2] = X[1] + X_{direction} \bullet X[2]_{travel\_distance} \qquad (19)$$
$$X[3] = CV_{xpresent\_location} \qquad (20)$$
$$Y[3] = CV_{ypresent\_location} \qquad (21)$$

Once the aircraft reaches the carrier or air base, it remains on the surface for a certain time interval (loiter time), which represents the refueling and maintenance of the aircraft, and rest for the crew. The loiter time for the aircraft is tracked by the same method used for naval ships. When the loiter time has elapsed, new waypoints are computed based on the location of the air base or present location of the aircraft carrier, and a reconnaissance aircraft commences flying its search pattern.

In practice, reconnaissance aircraft tend to follow courses that are judged to be potentially informative. The specific waypoints defined for the analysis performed in this thesis ensure that the courses are potentially informative. In a realistic environment, aircraft cueing, e.g. by an external (satellite) or shipboard sensor, is used to vector aircraft to a specific location. This feature is not represented in the model, but is a desired improvement. A reconnaissance aircraft also needs the ability to alter its flight path if a hostile aircraft or missile is detected near the plane. In this model the reconnaissance aircraft are susceptible to attack by fighter aircraft that defend the CVBG and the air base. Fighter and attack aircraft are considered weapons systems for the purpose of this model and are discussed later in this section. The algorithm that determines reconnaissance aircraft motion is described in Appendix C.

d. Satellites

Each satellite moves between two waypoints that are defined in the initial data base. The course that a satellite follows is computed using equation (1), and the speed is constant (no variance) and is also defined in the initial data base. Each satellite travels from its initial waypoint to the other waypoint. Once it reaches the new location, it loiters a set amount of time, which is intended to represent the time the satellite is passing over other regions of the earth during its orbit. Once the time is reached for the satellite to commence a pass over Iran, the satellite travels from its present location to the other waypoint. This simulates a back and forth pass

over the same region which is not correct, but is done as a programming simplification. For the purpose of this study, the waypoints are defined so that the satellites fly over areas of Iran where SSM launchers and air bases are located. The initial data base provides the U. S. forces with a pair of satellites. One of the satellites flies over Iran south to north, and one flies east to west.

2. Search and Detection

a. Assumptions

1. The ESM and ELINT sensors are only able to detect and localize targets that are emitting.

2. The air search sensor only detects airplanes.

3. The surface search sensor detects ships, SSM launchers, air bases, and radar sites.

4. This model does not take into account the altitude of the aircraft.

5. The U. S. satellites are undetectable by the Iranian sensors.

6. All detection information is immediately shared among all friendly units (i.e. communications links work perfectly: there is no congestion delay).

7. The U. S. assets always emit (radar in operation), and therefore are susceptible to detection by the Iranian ESM sensors. Iranian air bases and long range radar sites always emit, and therefore are susceptible to detection by U. S. ESM sensors.

All of these assumptions are candidates for modification in later revisions of the model.

b. Search and Detection Theory

Each sensor system has a maximum detection range expressed in the depth of surrounding grids throughout which there exists a positive probability of detecting an enemy unit. That is, if the range is one grid, detections occur within the grid at which the sensor is located, and all adjacent grids, (assuming a target susceptible to detection is present). The detection range assumes that the sensor is located in the center of the grid.

Each sensor has a probability of detection given a unit is within its detection range. Upon detection of an enemy unit, the perceived location of the enemy unit is reported by the sensor, which is the actual location (ground truth) offset by errors in the x and y direction. This perceived location is distributed as a circular normal

distribution with parameters μ_x, μ_y, and σ_{sensor}. The values of μ_x and μ_y are the ground truth x and y coordinates of the target, and σ_{sensor} which is a sensor error term accounting for the accuracy of the physical sensor and human operator of the system. The information derived from each sensor is assimilated with previous detections of the same unit to refine the intelligence estimate of enemy capabilities.

c. Search and Detection Implementation

On every time step (one minute), each unit searches for enemy units using all three of its sensor systems in order: surface, air, and ESM. For each sensor, the model determines the grids surrounding the unit where detections may occur given a target is present, and then sequentially searches each grid. Within each grid, the model individually examines each enemy unit located on the grid's list of enemy units and determines if it is eligible for detection by the sensor (i.e. aircraft by air search sensor, unit that is emitting by ESM sensor, etc.). Given it is a candidate, an independent uniform random variable with range [0,1] is drawn. If the random variable, which is denoted $H_{detection}$, is less than the probability that the sensor detects a target given it is within its search area, a detection occurs. If a target has been detected, the sensor reports the perceived location of the target as described previously. Equations (22) and (23) illustrate this approach.

$$H_{detection} \sim U(0,1)$$

| Detection occurs | if $H_{detection} \leq p_{det[sensor]}$ | |
| No detection | if $H_{detection} > p_{det[sensor]}$ | (22) |

$$\text{Perceived Location} \sim N(\mu_x, \mu_y, \sigma_{sensor}) \qquad (23)$$

This simulation assumes that all sensors perfectly determine the identity of a target (unit identification number) and there are no false detections.

d. Intelligence Estimate

As discussed earlier, enemy unit detections are maintained on lists that are resident on the grid system. Each force has 700 lists of detected enemy targets (a list for every grid). Some of the lists contain numerous targets, while others remain empty the entire simulation. These lists identify the units by a specific unit number, the current estimate of its Cartesian location, the number of detections of this enemy unit, and the time of the latest detection. Each time a sensor makes a new contact, i.e. achieves a detection, it determines the grid location corresponding to the perceived location reported by the sensor. The list located on that grid is then scanned to locate previous observations on this particular contact. If there is a contact history, an average

20

updating process is used to refine the contact location, otherwise the new contact is added to the list as a different entity. The perceived location of a target may be on a grid where the target is not actually located. If two separate detections locate the same unit, but the perceived locations are significantly different and place the units on two different grids, there will be an entry for the unit on each grid. In a real scenario, a person analyzing this information may believe there are two separate contacts. However, the weapon targeting model of this simulation does not allow the firing of two separate sorties of weapons at the same target based on entries on two different grid lists.

Equations (24) through (26) describe the information fusion process. The variables $X_{perception\_old}$, $Y_{perception\_old}$, and number of observations are defined for each unit that is located on a grid's list. If a unit is located on two separate grid lists, it has a set of three variables for each list. The information contained within those two sets of variables which are defined on different numbered grids is never combined to provide a more precise estimate of a unit's actual location. The $X_{perception\_old}$ and $Y_{perception\_old}$ variables are the average X and Y perceived locations over all detections of a unit which place it on a particular grid, but does not include the most recent detection. The number of observations is the number of times the unit was reported on this grid prior to the most recent detection. The $X_{perception\_new}$ and $Y_{perception\_new}$ variables are the average perceived X and Y location of the unit after they have been averaged with the most recent detection. The number of observations is incremented to indicate incorporation of this new detection in computing the unit's average perceived location.

$$X_{perception\_new} = (\#observations \bullet X_{perception\_old} + X_{perception\_new})/(\#observations + 1) \quad (24)$$
$$Y_{perception\_new} = (\#observations \bullet Y_{perception\_old} + Y_{perception\_new})/(\#observations + 1) \quad (25)$$
$$\#observations = \#observations + 1 \quad (26)$$

The model deletes a contact position on a specific grid if the contact has not been reported as detected on that grid by at least one sensor among the entire force, during the latest T update cycles ($T = 15$ minutes). This is done to eliminate contacts on the targets that have changed grids, were reported incorrectly as on a grid, or have been killed. The model does not allow a sensor to detect a target that has been killed.

3. Weapons Engagements

a. Engagement Theory

Each weapon system is assigned probability of defeat by the target's defensive mechanisms (p_{defeat}) and a probability of hit (p_{hit}) against a particular target type by the user prior to a simulation run. The defensive capability of each unit against a type of weapon is aggregated into a single number (p_{defeat}) that represents the probability that the unit will defeat an incoming weapon prior to the weapon releasing its ordnance. Each unit has is assigned a different probability of defeating an incoming weapon for each type of weapon that it may encounter. Defensive weapons systems such as Iranian land-based surface-to-air missile (SAM) sites, which protect high-value targets (air bases, radar sites) are included in this aggregation. The only time an incoming weapon is susceptible to destruction is when it reaches its target.

In a real conflict, the probability that a unit can defend itself is affected by previous damage incurred by the unit. The computer implementation of this simulation model assumes a unit's defensive capabilities are fully operational until it is destroyed. This model could be improved by determining the strength of a unit's defensive capabilities based on the number of hits it has previously received and decreasing the probability that its defensive mechanisms defeat an incoming weapon (p_{defeat}). Additionally, in the case of naval ships operating as a task group, the damage to the group as a combined entity, affects the ability of each unit to defeat incoming weapons. Under wartime conditions, the Ticonderoga class cruisers are responsible for the air defense of the carrier battle group. If the cruiser is damaged then the ability of all other units to defend themselves against air attacks is diminished. This feature should also be addressed in future improvements of the model.

The determination of whether a weapon scores a hit against a target depends on its probability of hit (p_{hit}) against that target type and the accuracy of the targeting information at the time of fire. In this model the probability of hit represents all weapon delivery error (wind, targeting computer, etc.). The probability of hit is reduced due to target location errors. A targeting model, which is discussed later, designates a sortie, which is a specific weapon type and the number of weapons of this type to fire at a target. The sortie is targeted at the location where the aggressor believes the enemy is positioned ($X_{perceived}, Y_{perceived}$). The distance between this position and the ground truth location of the target (X_{actual}, Y_{actual}) is used to compute a multiplier (D_{hit}), which degrades the probability (P_{hit}) that a weapon scores a hit against the target (Phit)Phit. This simple approach, which is shown in equations (27) and (28),

describes how the accuracy of the perceived target's position influences the probability that the weapon scores a hit against a target. Since it computes the aimpoint error at the time of fire, it does not enable the weapon to receive mid-course guidance, or model weapons that can manually adjust their course, such as piloted aircraft and radar-seeking seeking missiles.

$$AimptError = \frac{\sqrt{(Xactual - Xperceived)^2 + (Yactual - Yperceived)^2}}{10} \qquad (27)$$

$$D_{hit} = \begin{matrix} e^{-AimptError} & \text{if Aimpoint Error} \leq 10 \text{ miles} \\ .2 & \text{if Aimpoint Error} > 10 \text{ miles} \end{matrix} \qquad (28)$$

Until the sortie reaches the target and the result of the strike is determined, no other sorties can be fired at the target. This does not allow for a force to utilize the tactic of saturating the enemy with incoming weapons. When employed, this tactic increases the probability of some of the weapons evading a target's defensive mechanisms and striking the target. The weapon selection decision module does not select a mixed group of weapon types to fire at a single target.

The simulation time at which the sortie is to arrive at the target's location is computed using the distance between the sortie's point of origin(X_{firing_unit}, Y_{firing_unit}) and the ground truth location of the target(X_{actual}, Y_{actual}), the nominal (operational) speed of the weapon type (*Weapon speed*) which comprises the sortie, and a preparation time. The preparation time represents the time required to compute a firing solution and prepare weapons. This time is referred to as the on target time (OTT) and is computed using the equation (29).

$$OTT = Present\,Time + Prep\,Time + \frac{\sqrt{(Xfiring\_unit - Xactual)^2 + (Yfiring\_unit - Yactual)^2}}{Weapon\,speed} \qquad (29)$$

This assumes that the target does not change location during the sortie transit time, which is not realistic but is done for simplicity of modeling. The flight path of missiles and fighter and attack aircraft are not placed on the grid system. This is a simplification of the model and ignores the intelligence gained by placing it on the grid. An attacking aircraft provides additional reconnaissance during its flight, and the direction of an incoming strike gives the target a rough idea of the location of the platform that launched the strike.

Once the simulation time is reached that indicates a sortie has arrived at the target's location, an adjudication of the attack is conducted. During conflict adjudication, the model first determines how many of the weapons in the sortie evaded the target's defensive mechanisms (p_{defeat}). During a real strike, the direction and time that multiple weapons arrive at a target are not always the same. For example, a group of attacking aircraft may separate on their final approach to a target and attack from different directions and in waves to confuse and complicate the defensive response of the target. As a result, in this simulation, the determination of whether a single weapon of a sortie evaded the target's defensive mechanisms is considered independently of the sortie's other weapons. Hence, the number of weapons that evade the target's defensive mechanisms has a binomial distribution with probability of success equal to p_{defeat} and number of trials equal to the number of weapons in the sortie. The number of hits scored against the target by those weapons that get through the defensive mechanisms is determined. The number of hits has a binomial distribution with probability of success equal to $P_{hit} \bullet D_{hit}$ and number of trials equal to the number of weapons that survived the target's defensive mechanisms.

Each unit has a counter that keeps track of the total number of hits it has sustained during the course of the simulation. The total number of hits the target received during this strike is added to the counter, and a check is performed to ensure the target has not exceeded its maximum number of hits. A target that has sustained at least its maximum number of hits is considered destroyed. If the target is destroyed, its type is changed to indicate it no longer exists. This means that its sensor, weapon and propulsion (movement) systems are rendered non-functional and provide no input for its respective force the remainder of the simulation. If the attacking sortie consists of aircraft, the number of the aircraft that are not destroyed by the target's defensive mechanisms are returned as available assets onboard the carrier or air base after a time delay which accounts for their return flight.

b. Engagement Implementation

(1) **Overview.** The model maintains two engagement lists for conflicts between the two forces, one for the United States force striking Iranian units, and another for an Iranian force striking U. S. units. During every update cycle, the targeting module examines every grid's intelligence estimate list sequentially,

starting at grid one and finishing at grid 700. Three steps are performed in this examination phase for each enemy unit on a grid list that is not currently targeted (i.e. a weapon is not enroute to the target) or dead, which determines the action to be taken against an enemy unit. The first step classifies the threat level of the target, and determines the closest missile-capable and aircraft-capable friendly unit that has weapons available to engage the target. In the next step, one of the two units is assigned to prosecute the target using a sortie. The decision of which unit to assign is based on the target type and is described by the targeting algorithms included in Appendix C. Finally, the time for the sortie to reach the enemy unit (time of conflict adjudication) is computed, and the conflict placed on the engagement list. It is important to note that these three steps are performed on each target individually before addressing the next target. Also all contacts on the grid one list are examined and decisions made on weapons to fire at all of the targets listed as on grid one prior to examining targets that are listed as on grid two, and so forth. A result of this process all targets are prosecuted, whether high or low threat until weapons supplies are depleted starting at the lowest numbered grid. If a high threat target is located on a grid with a high grid number, the possibility exists that weapons will not be available to prosecute the target. In future revisions of this model it is recommended that a more realistic weapons allocation procedure be implemented. Once all targets which exist in the entire grid system have been placed on this list, a determination on which targets to prosecute can be made on the basis of threat level.

(2) **Threat Determination and Closest-Unit Selection.** During this step, the enemy unit is classified as a high, medium, or low threat, in accordance with the algorithm located in Appendix A. The algorithm first computes the distances between the perceived enemy position ($X_{perceived}$, $Y_{perceived}$) and every one of the friendly units. If any of the distances is smaller than the effective weapon range (of a weapon that can attack a given friendly unit) of the enemy unit, it is classified as a high threat; otherwise it is a low threat. There are two exceptions; an Iranian long-range radar site is classified as a medium threat, and an Iranian air base which is located where its aircraft are not within the range that they can strike any multinational force or U. S. naval ships is classified a medium threat instead of a low threat. During this step the distance information is used to identify the closest friendly missile-capable unit and aircraft-capable unit that has both weapons available

(#weapons > minimum reserve number of a weapon that must remain onboard a unit at all times), and is located within effective weapon's strike range of the enemy unit for possible target prosecution.

(3) **Assign Unit and Allocate Weapons.** There is a separate decision module for each force (U. S. and Iranian) which determines which unit prosecutes a given enemy unit, and the numbers and type of weapon to utilize. Both of these decision modules are somewhat arbitrary and do not follow any known military guidance. Alternative decision processes can be reasonably contemplated and studied within the structure of this simulation. The decision module bases its unit selection on the level of threat, and the type of enemy unit. For example, a high threat Iranian air base will be assigned to an aircraft carrier using numerous attack aircraft, whereas, a low-threat Iranian air base would be assigned to the closer of a Tomahawk-capable warship or a carrier dispatching a smaller number of attack aircraft. The two algorithms that determine U. S. and Iranian weapons allocation are located in Appendix C. Once the unit and weapon assets are determined, the counters that track unit weapon availability and the force weapon use are updated accordingly. In the case of attack or fighter aircraft, the carrier or Iranian air base will not gain the availability of those assets until the strike is completed (conflict adjudicated) and aircraft fly back to their point of origination.

(4) **Engagement List.** There is a separate engagement list for each force (U. S. attacking Iranian and its counterpart). The time the weapon is on target and aimpoint error degradation (D_{hit}) is computed as stated previously in the engagement theory discussion. The engagement is then added to the engagement list. Each item on the list identifies the time the weapon reaches the target, the unit that fired the weapon, the enemy unit that is under attack from this weapon, the type of weapon, the number of weapons fired, and the aimpoint error. Once a conflict is adjudicated it is removed from its engagement list. This entire targeting process may be improved by developing more detailed threat prioritization rules, and refining the rules to assign weapons to a target based on current military doctrine.

c. Conflict Adjudication

(1) **Defensive Weapons.** When a sortie reaches its intended target, independent Bernoulli trials are simulated for each weapon of the sortie in accordance with equation (30) to determine if the number of weapons that survive the target's defensive mechanisms. H_{defeat} is a draw from a uniform distribution over [0,1].

$$H_{defeat} \sim U(0,1)$$

Weapon survived   if $H_{defeat} > p_{defeat}$
Weapon destroyed   if $H_{defeat} \leq p_{defeat}$ (30)

The independent assumption is used since weapons fired at a single target are often fired at different times and may approach the target from varied directions. If the weapon is not destroyed, it then has a probability of scoring a hit against the target.

This representation of the defensive weapon systems is simplistic in nature, and further improvements should model significant defensive systems as separate entities. Another improvement to the model would allow the engagement of incoming weapons prior to reaching their target. In a real conflict, U. S. strike aircraft would have to avoid Iranian SAM sites located just inside Iran's borders, and Iranian strike aircraft could be engaged by U. S. combat air patrol aircraft and surface-to-air missiles from U. S. warships long before they reach their intended target.

(2) **Offensive Weapons.** Once the number of weapons that survive the target's defense mechanisms is computed, the number of hits scored against the target is determined in accordance with equation (31). Independent Bernoulli trials are conducted for each weapon striking the target to determine how many hits the target sustained. H_{hit} is a draw from a uniform distribution over [0,1].

$$H_{hit} \sim U(0,1)$$

Weapon hit target   if $H_{hit} < p_{hit} \bullet D_{hit}$
Weapon missed target   if $H_{hit} \geq p_{hit} \bullet D_{hit}$ (31) .

Whenever an aircraft carrier or air base is a target, it is assumed that both the area where planes are launched (runway) and the location of planes on the surface are targeted. This model does not possess the detail required to adequately account for the attrition of planes under these circumstances,

27

however, a simple approach is implemented which recognizes the attrition of planes. A number is defined, $Ac_{weapon\ type}$, which attempts to represent both the location of aircraft throughout an air field, or aircraft carrier, and the lethal area of an incoming missile or bomb. These numbers for U. S. weapons against an Iranian air base and Iranian weapons against a carrier are defined at the beginning of the simulation. This number does not address airplanes that are grouped together in the same area, so an additional random number is computed called the effectiveness (E) of the strike to identify this situation; E is a uniform random variable over [0,2]. The range of the uniform random variable was arbitrarily set between zero and two for analysis purposes. The total number of aircraft killed is computed using equation (33). This equation uses the number of aircraft that are located on the air base or carrier at the time the conflict is adjudicated. Since both attack and fighter aircraft may be destroyed at a location, the model divides the number of aircraft killed in proportion to the number of each aircraft that is located at the base or on the carrier. If the number of fighter or attack aircraft killed exceeds the number of aircraft at a location, the maximum number of that kind of aircraft is destroyed. This approach is illustrated using equations (32) through (35).

$$E \sim U(0,2) \tag{32}$$

$$\#Aircraft\ Killed = \#hits \bullet Acweapontype \bullet \#Aircraft \bullet E \tag{33}$$

$$Fighter\ Killed = \left(\frac{\#FighterAircraft}{\#Aircraft}\right)\#AircraftKilled \tag{34}$$

$$Attack\ Aircraft = \left(\frac{\#AttackAircraft}{\#Aircraft}\right)\#AircraftKilled \tag{35}$$

28

IV. SIMULATION

A. OVERVIEW

The simulation consists of the model, variables that record aggregate weapons and unit information, and decision modules that force the units within the model to interact over the course of a multiple-day battle. The simulation occurs in cyclic updates; it is performed at one-minute intervals of cycle lengths, but is easily modified to change the length of cycle. The simulation is programmed in the C++ language to perform quantitative analysis.

There are three different sources of information which affect the results of the simulation. The three sources are the initial database (geographic and force structures), the decision modules that govern force interactions, and the decision parameters utilized in the decision modules. All three of these are fixed prior to conducting a simulation run. This thesis concentrates on studying the effects of adjusting the force structure and characteristics of individual units in the initial database.

The output of the simulation consists of three files that track specific parameters over the course of the simulation. The output files are used to keep track of the locations of all units during the simulation, damage incurred by individual units, the total number of weapons fired, weapons destroyed, and casualties incurred by all forces at the end of the two-day battle.

B. UPDATE CYCLE

The update cycle is diagrammed in Figure 6. The initialization phase, which loads the geographic, U. S., and Iranian force structure database, and initializes all counters, is conducted at the beginning of a simulation run. During a typical cycle, each unit loiters or continues its transit towards its next waypoint. Each unit then uses all of its available sensors to detect enemy contacts. After each force processes all the information received from its sensors, a refined intelligence estimate is drawn. Each force then determines the enemy units that it considers threats, and commits units to use their weapon systems to strike enemy units. If any weapons have reached their intended target, the resulting conflict is adjudicated. If the time is reached for an aircraft to return to its point of origin, the number of available air assets for its carrier or air base is updated to reflect its return. The clock is then incremented and the process continues.

29
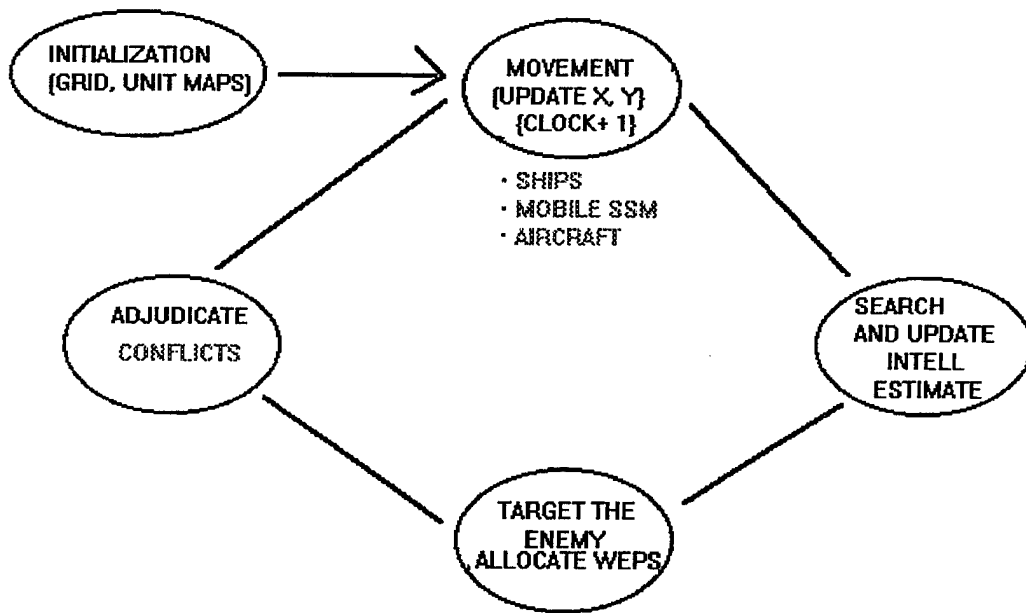
Figure 6. Simulation Update Cycle

1. Initialization

The initialization phase of the simulation loads the grid information onto the map from a data file. The simulation then reads the list of units from a data file into the array. The simulation then examines the initial location of each unit to determine the grid where each unit is located, and adds each unit onto the list for its respective grid. Initially, all units are stationary and will change position according to the movement algorithms. There is presently no initial intelligence estimate. A separate data file (resulting from random initial sightings of enemy units) may be incorporated later as an initial intelligence estimate.

2. Clock

The simulation performs an update cycle at one-minute intervals of real time. This implies that each unit's position will change using the distance it covered over the last minute of time. The one-minute interval is chosen to accommodate aircraft and missiles which move at high speed.

3. Movement

This phase of the simulation determines whether a unit is in motion, or should start movement. If it is in motion it computes its present location and checks to see if it moved to a new grid. If it has moved to a new

grid it updates the associated grids to reflect this change. If motion starts on this time step, it updates the unit to show that it is in motion and determines the time at which the unit will reach its next waypoint. All motion is conducted according to the algorithms located in Appendix A. The movement of all units is updated during every cycle which equates to one minute of scenario time.

4. Sensor Update

Every sensor in the simulation examines all the grids within its maximum detection range during each time step. The intelligence estimate is then updated accordingly. If there is no detection of a contact for the last T minutes of simulation time, the contact will be dropped from the intelligence estimate. The satellite assets, which are an important factor in this simulation, have a behavioral artificiality: each scans over a collection of grids (a region of the Iranian country), but during the next pass of the satellite over the same region, it scans the grids in reverse order. This is a computer artificiality because of the method used to simulate traveling between waypoints.

5. Enemy Targeting

The model generates the two engagement lists and updates global variables which track total force weapon use. Whenever an enemy unit is targeted, the targeted variable of the unit is flagged to indicate a weapon is enroute, which prevents multiple weapons fired at the unit at the same time. The simulation does not model the situation in which commanders desire to saturate one target with multiple types and numbers of weapons.

6. Conflict Adjudication

During this step, the simulation examines each conflict on the engagement list and determines if the weapon is now over the target. All the engagements that meet this criterion are removed from the engagement list and adjudicated. After the conflict is adjudicated, the model updates variables to indicate weapons and units destroyed, and places any surviving attacking aircraft on the time delay list. If a unit survived the assault, its targeted variable is unflagged, and it may now be fired at again.

After all the conflicts for a given time step have been adjudicated, the simulation examines the list of aircraft that are flying back to aircraft carriers and air bases. All attack and fighter aircraft that have completed their return flight during this time step are added as available assets to their respective carrier or air base.

C. SOURCES OF INFORMATION

As previously discussed, the simulation is influenced by three different sources. The first is the algorithms and their decision parameters (numbers of asset to use in a strike attack, the number of weapons a unit must maintain in reserve, etc.) that define the decision modules utilized in the simulation. They specify the doctrine under which the battle is conducted, which remains unchanged for the analysis of this thesis. The algorithms that were used to realize the simulation as a computer program are included in Appendix C. The second influence is the list of constants related to sensor and weapons effectiveness probabilities that are used in the decision modules. The constants specify probabilities (P_{detect}, P_{defeat}, and P_{hit}), sensor accuracy, weapons speed, and lethality of weapons. These parameters are constant for a given set of simulation runs and are listed Appendix D. The third source, is the initial characteristics of each unit, and the numbers of units that are loaded into the simulation. The initial characteristics include weapon types, numbers, and ranges, unit operating speeds; intended track; number of hits a unit can withstand and remain functional; sensor range; and emissions control status. This information is maintained in the initial force database and is the easiest source of information to vary.

D. COMPUTER MODEL

The simulation is coded using an object-orientated approach in the programming language C++ which is included as Appendix E. Many of the programming techniques and data structures were drawn from *C++ How to Program* (Deitel, 1994). The data structure which stores the characteristic of each grid and unit defined by the initial database, and the decision module algorithms are hard coded into the simulation. The parameters which are utilized by the algorithms are also hard coded into the simulation, but are grouped together at the beginning of the main source code file and are easily modified by anyone familiar with general programming practices. The geographic and force structure databases were built in one file using a spreadsheet. This file is easily modified by a user familiar with the operation of spreadsheets, and requires no modification of the computer code.

E. OUTPUT FILES

1. Unit movement (Motion.dat)

This file is updated during every time step and records the Cartesian location of each unit along with its unit identification number. This file is used to verify the units are moving correctly in accordance with both the initial database parameters and movement algorithms.

2. Unit losses (Loss.dat)

This file is updated each time a unit is killed. The simulation records the unit identification number and the simulation time at which it was killed.

3. Unit status (Unit.dat)

This file is the primary data source for measures of effectiveness (MOE) calculations and evaluation of the battle. This file contains a complete listing of the characteristics of each unit at the beginning of the simulation (t = 0), and the completion of the simulation. The information includes the number of hits suffered by each unit, and number of weapons left over after the battle. The first list is used in verifying that the simulation correctly input the data from the initial database. The second list provides information for measure of effectiveness (MOE) calculations. At the end of the simulation, the program also outputs to this file the variables that keep track of the number missiles fired, attack and fighter aircraft launched, units killed, and aircraft lost on strike or interdiction missions.

V. EXPERIMENTATION AND ANALYSIS

A. OVERVIEW

The output of this simulation is a function of numerous input parameters. The goal of this thesis is to construct and exercise a proposed simulation model that has the capability to examine the effect of varying conditions that relate to the sensors and weapons available to the units that comprise a United States carrier battle group. A factorial experiment was used to study one instance of the conflict described in this thesis to examine how changing three of the initial conditions affect the outcome of the battle. The *Minitab*® *Release 11 for Windows* statistical software package, whose capabilities are described in the *Minitab* ® *Reference Manual Release 11 for Windows* (Minitab, 1996), is used to demonstrate techniques for analyzing the output from this simulation.

The result of the conflict is summarized using seven measures of effectiveness (MOE), which judge the outcome of simulation trials with respect to the success of the United States and multinational forces. Output is gathered by performing computer trials of the simulation; this information is used to calculate the seven MOE. A 2^3 full factorial experiment using ten replications of the computer simulation for each factor level was conducted and analysis performed on the output (specific parameters of the initial data base and program are modified as required to vary the conditions to conduct the experiment).

B. MEASURES OF EFFECTIVENESS

The measures of effectiveness are defined to represent the ability of the U. S. force to accomplish its mission while minimizing damage to its own forces, and to describe the results in terms of the United States combat effectiveness and survivability. The measures of effectiveness do not measure the attrition rate of attacking weapons since they depend only on the probability of defeat (p_{defeat}) by a target's defensive mechanisms in the computer implementation of the simulation. The following MOE are defined for the model:

1. Total number of hits sustained among all ships comprising the carrier battle group over the two-day battle.

2. Total number of hits sustained among all ships comprising the multinational force over the two-day battle.

3. Fraction of CVBG ships remaining at the conclusion of the two-day battle.

4. Number of multinational force ships remaining at the conclusion of the two-day battle.

5. Combined number of U. S. missiles and attack aircraft launched against Iranian targets over the two-day battle.

6. Total Number of hits scored against all Iranian land-based targets over the two-day battle.

7. Total number of Iranian attack aircraft launched against U. S. ships over the two-day battle.

C. EXPERIMENTATION

The 2^3 factorial experiment varied three conditions which are the: (1) The availability of satellite intelligence to the U. S. forces; (2) U. S. sensor accuracy; and (3) number of U. S. aircraft carriers. The United States either receives satellite information from one pair of satellites, or it receives no satellite information the entire simulation. Upon detection, a sensor reports the perceived location of the target which is the actual location offset by a normal error. For one level of the sensor accuracy factor, the normal errors for all sensors have mean zero and variance of 16 miles for surface and ESM sensors and .25 mile for air sensors. For the other level of sensor accuracy, the normal errors of the United States and multinational force sensors errors have mean zero and variance zero and so the target is located perfectly; Iranian sensor errors have mean zero and variance of 16 miles for surface and ESM sensors and .25 mile for air sensors. Hence, the sensor accuracy factor eliminates the sensor error and causes all U. S. and multinational force sensors to provide perfect information on the location of a target. The U. S. operates with one or two aircraft carriers in the battle group. The addition of a second aircraft carrier provides a second air wing with its attack, fighter, and reconnaissance aircraft. The design of the experiment and the analysis techniques that are applied to develop conclusions are described in *Statistics for Experimenters, An Introduction to Design, Data Analysis, and Model Building* (Box, 1978, pp. 306 - 321). The experimental runs are arranged in the standard order as shown in Tables 1 and 2.

Table 1. Factor Levels for 2^3 Factorial Experiment

Variable	-	+
1. Satellites:	0	2
2. Sensor accuracy:	Normal errors	Perfect information
3. Aircraft Carriers :	1	2

Table 2. 2^3 Factorial Design, Trials Arranged in the Standard Order

Trial Number	Satellites	factor in simulation Sensor accuracy	Carriers
1	0	Normal errors	1
2	2	Normal errors	1
3	0	Perfect information	1
4	2	Perfect information	1
5	0	Normal errors	2
6	2	Normal errors	2
7	0	Perfect information	2
8	2	Perfect information	2

D. ANALYSIS

1. Overview

The full 2^3 factorial experiment was performed with ten replications performed for each factor level. Each of the MOE computed from the output of the simulation runs is examined using statistical analysis tools. The experiment results are included as Appendix F.

2. Analysis Techniques

The first step in exploring how the initial conditions affected the outcome was to graphically explore the results of the experimentation. The individual data points were plotted and both these graphs and the data are included as Appendix F. The plots were examined to note significant trends and determine what analysis techniques to apply for each MOE.

If the plot of the observations of an MOE leads one to believe that the observational distribution is normal, a fractional factorial analysis fit was computed for the MOE using the *Minitab*® statistical software package. This analysis first computes a fractional factorial estimate of how each factor individually (main effect), and when combined with other factor(s) (multiple-factor interaction effect) affects the MOE. The result of this analysis provides an estimate of how each effect changes the MOE and a probability value (p-value),

37

which is the probability of getting a test statistic value at least as extreme as the one actually obtained under the null hypothesis that there is no effect. If the p-value is less than a desired significance level α, the effect may be significant and requires interpretation. The main effect of a variable should be individually interpreted only if there is no evidence that the variable interacts with other variables. When there is evidence of one or more such interaction effects, the interacting variables should be considered jointly, and other analysis techniques used to further explore the main effects. In using this analysis technique, first look at the three-factor interaction and decide if it is significant. If the three-factor interaction is significant, interpretation of main effects and two-factor interaction effects are not valid on the basis of this analysis technique alone. If there is no three-factor interaction, but one or more two-factor interactions, only the two-factor interactions are interpreted and no conclusions can be drawn about the main effects. (Box, 1978, pp. 317-318).

The software performs a three-factor analysis of variance (ANOVA), which is a technique that determines whether the different effect estimates differ from zero more than could reasonably be expected under the null hypothesis of no effect. The ANOVA technique tests the null hypothesis that the means of the effects for each level of interaction are equal to zero, against the alternative hypothesis that there is at least one that is not zero. The software computes an F-ratio and a p-value for each level of interaction (main, two-way, three-way). This p-value is the probability of obtaining a realization from the F-distribution under the null hypothesis of no effect at least as large as the observed F-ratio. The theory behind the computation and use of the F-ratio and p-value in the ANOVA analysis can be found in *Probability and Statistics for Engineering and the Sciences* (Devore, 1995, pp. 444-452).

A second procedure, the Kruskal-Wallis test, which tests whether an arbitrary number of k independent populations are identical is also used, and is applicable to all the MOE results regardless of their distribution. This test is similar in spirit to one-way ANOVA analysis and tests the null hypothesis that the k populations are identical, against the alternative hypothesis that there are at least two of the populations that are different. The Kruskal-Wallis test computes an H-value and compares it to a χ^2 distribution to compute a p-value. If the p-value is so small that the null hypothesis is not supported, it is concluded that the compared populations are not all the same. A theoretical development and explanation of this test is found in *Nonparametric*

Methods for Quantitative Analysis (Gibbons, 1985, pp. 173-179). The *Minitab*® Kruskal-Wallis test that is

adjusted for ties was used to obtain an *H*-value and a p-value for each factor.

The *Minitab*® statistical software package was then used to develop cube plots of the means for each

MOE and compute basic summary statistics (mean, standard error of the mean, and standard deviation) of each

individual trial for all MOE.

3. MOE 1

Since the data points for MOE 1 appear normally distributed, as shown in the plot included in

Appendix F, a factorial analysis was computed and is shown in Table 3. For any effect with a p-value less

than .05 (α = .05 significance level), the null hypothesis that the mean of the effect is equal to zero is

rejected and the alternative hypothesis that the mean is different from zero is accepted; hence, the effect is

considered significant. The use of this significance level is consistent throughout all analyses that are

discussed in this thesis. The ANOVA table indicates that there is at least one strong two-way interaction,

with a corresponding p-value of .03. Due to the strong two-way interaction which occurs between the use

Table 3. Analysis of Variance Table for MOE 1 with Estimated Effects

Source	df	SS	MS	F-ratio	p-value
Main effects	3	5145.40	1715.10	12.37	0.00
Two-way interactions	3	1343.30	447.80	3.23	0.03
Three-way interaction	1	515.10	515.10	3.71	0.06
Residual error	72	9986.50	138.70		
Total	79	16990.40			

Factor	Effect	p-value
mean	28.30	-
satellite	-0.43	0.87
sensor	-10.38	0.00
carrier	12.23	0.00
satellite & sensor	3.83	0.15
satellite & carrier	5.93	0.03
sensor & carrier	4.18	0.12
all	5.08	0.06

Note: The value for the mean effect is the overall mean value of MOE 1 for the eighty
simulation runs. The values for the other effects represent the estimated change in MOE 1 when an
individual factor or combination of factors are at their high level (+) as compared to the when all of the
factors are at their low levels (-).The p-value of the estimated effect is the probability that the value for the
effect is actually zero.

of satellite intelligence and an additional carrier (p-value = .03), the large main effects of perfect sensor information or adding an additional carrier cannot be interpreted based on this analysis technique. Before accepting the results of the factorial analysis, a normal probability plot of the residuals is performed to verify the normality assumption used to enter into this analysis. The plot of the residuals in Figure 7, shows that the normality assumption is probably reasonable, and the results of this analysis technique are valid.

Normal Probability Plot of the Residuals
(response is moe 1)
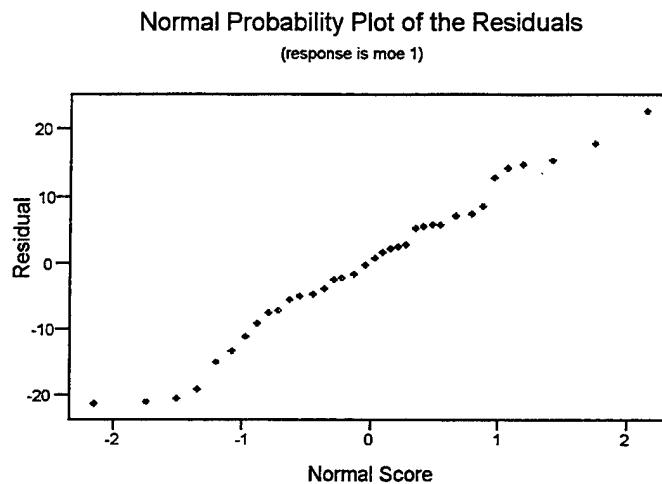
Figure 7. Normal Probability Plot of the Residuals for the Fitted Factorial Model of MOE 1

The most information is obtained about this MOE by examining a cube plot of the means, which is shown in Figure 8. A comparison of the means for the two levels of sensor accuracy clearly shows that when perfect sensor information is used, the mean number of hits sustained decreases in all cases except when an additional carrier is added to the battle group. If the means of the two levels for the number of carriers in theater are compared, it is noted that the mean number of hits increases with the addition of a second carrier, regardless of the changes in the other two factors in all but one case. The exception is when there are two carriers, perfect sensor information, and no satellite intelligence. No reasonable explanation is offered for this exception, and it requires further study. The only conclusion that can be drawn is that using perfect sensor information when operating with one carrier causes the mean number of hits sustained by the ships among the carrier battle group to decrease.

Cube Plot - Means for MOE 1
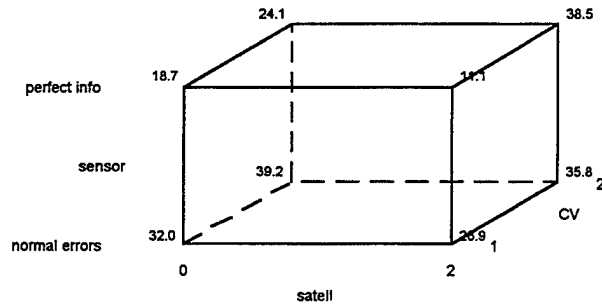
satellites	sensor accuracy	aircraft carriers	mean	se mean	std dev
0	normal errors	1	32	3.5	11.06
2	normal errors	1	26.9	4.19	13.25
0	perfect information	1	18.7	4.37	13.82
2	perfect information	1	11.1	4.33	13.68
0	normal errors	2	39.2	4.29	13.56
2	normal errors	2	35.8	2.62	8.28
0	perfect information	2	24.1	3.61	11.42
2	perfect information	2	38.5	2.26	7.14

How to Read: The main effect of each factor is seen to be a difference between two averages, half of the eight results being included in one average and half in the other. The main effects may be viewed as a contrast between observations on parallel faces of the cube plot. Similarly, each two-factor interaction is viewed as a contrast between results on two diagonal planes. The mean values displayed on the cube are also listed in tabular form along with their standard error of the mean and standard deviation.

Figure 8. Means of the Total Number of Hits Sustained Among All Ships Comprising the CVBG Over the Two-Day Battle and Summary Statistics.

4. MOE 2

The number of hits sustained by the ships comprising the multinational force varies between two numbers and does not appear normally distributed, so an ANOVA analysis was not attempted. Each ship comprising this group has a maximum number of hits it can sustain before it is considered killed. The combined total hits for the four ships is 36, and in 27 out of 80 simulation runs all ships are killed. Since a sortie may contain numerous weapons, it may score more hits than needed to destroy a ship. This means the number of hits sustained may be greater than the maximum required to kill a ship. In two

cases, the mean number of ships killed is greater than 36. When satellite intelligence and an additional carrier is in theater, the mean number of hits sustained by the multinational force is estimated as 20 from the data listed in Figure 9. This suggests that some of the ships are not killed.

A Kruskal-Wallis test is performed for each individual factor. Each test compared the 40 observations of MOE 2 for a factor at its initial level (-) with the 40 observations at its optimal (+) level. The results of this analysis are summarized in Table 4. Since the p-values for using satellite intelligence and an additional carrier are less than 0.05 they are considered significant. This suggests that using satellite intelligence or an additional carrier causes the number of hits sustained by the multinational force to decrease.

Table 4. Kruskal-Wallis Analysis for MOE 2

factor	H-statistic	p-value
satellite	15.52	0.00
sensor	0.19	0.66
carrier	13.12	0.00

Comparing the mean number of hits sustained for each case, it is determined that only when both an additional carrier and satellite intelligence is available does the mean number of hits sustained by the multinational force decrease, regardless of sensor accuracy. Since the ANOVA table indicates that a two-way interaction appears to be present, the results of the Kruskal-Wallis test should be viewed with caution.

Cube Plot - Means for MOE 2
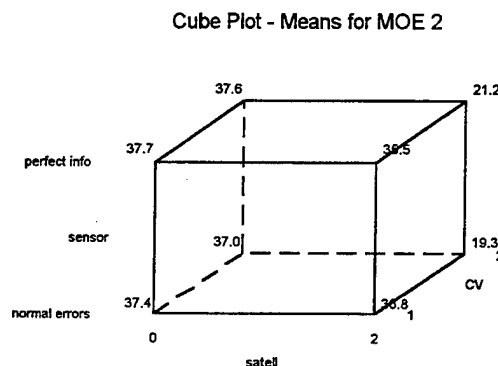
satellites	sensor accuracy	aircraft carriers	mean	se mean	std dev
0	normal errors	1	37.4	0.31	0.97
2	normal errors	1	36.8	0.952	3.01
0	perfect information	1	37.7	0.5	1.57
2	perfect information	1	38.5	1.26	3.98
0	normal errors	2	37	0.3	0.94
2	normal errors	2	19.3	2.44	7.7
0	perfect information	2	37.6	0.37	1.17
2	perfect information	2	21.2	2.62	8.3

Figure 9. Means of the Total Number of Hits Sustained Among All Ships Comprising the Multinational Force over the Two-Day Battle and Summary Statistics.

The only conclusion that can be stated is that the mean number of hits decrease when the U. S. forces utilize both satellite intelligence and an additional carrier.

5. MOE 3

Only a few ships among the seven comprising the carrier battle group are defeated during the two-day battle, and the most ships lost during any of the 80 simulation runs is two. This data is not normal and only the Kruskal-Wallis test is appropriate and the results are shown in Table 5. The results of the Kruskal-Wallis tests indicate that the use of perfect sensor information is significant. Examining Figure 10 shows that the mean fraction of ships that survive increases when perfect sensor information is used in all cases except when there are two carriers. These data points are so clustered, that any difference in the means may be due to normal variations as indicated by the summary statistics. No conclusions can be drawn for this MOE from this analysis.

Cube Plot - Means for MOE 3

43
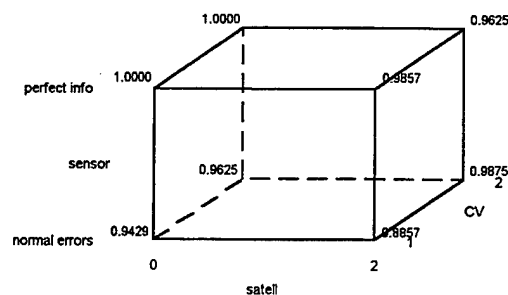
satellites	sensor accuracy	aircraft carriers	mean	se mean	std dev
0	normal errors	1	0.9429	0.032	0.01
2	normal errors	1	0.8857	0.036	0.11
0	perfect information	1	1	0.000	0.00
2	perfect information	1	0.9857	0.010	0.05
0	normal errors	2	0.9625	0.030	0.08
2	normal errors	2	0.9875	0.010	0.04
0	perfect information	2	1	0.000	0.00
2	perfect information	2	0.9625	0.190	0.06

Figure 10. Means of the Fraction of CVBG Ships Remaining at the Conclusion of the Two-Day Battle and Summary Statistics.

Table 5. Kruskal-Wallis Analysis for MOE 3

factor	H-statistic	p-value
satellite	2.45	0.12
sensor	5.89	0.02
carrier	2.02	0.16

6. MOE 4

The number of multinational force ships that survive the battle is directly related to the number of hits that the ships sustain, and there is a strong correlation between MOE 2 and MOE 4. Again ANOVA analysis is not appropriate, and Kruskal-Wallis tests are performed. The tests, which are summarized in Table 6, indicate that both the use of satellite intelligence and an additional aircraft are significant. The cube plot of the means clearly shows similar findings to those for MOE 2; when satellite intelligence is available along with a second carrier the mean number of multinational force ships that

Table 6. Kruskal-Wallis Analysis for MOE 4

factor	H-statistic	p-value
satellite	30.55	0.00
sensor	0.39	0.53
carrier	13.49	0.00

survive the two-day battle increases. This is consistent with MOE 2 where the use of satellite intelligence with two carriers caused the mean number of hits sustained by the multinational force to decrease. Since a two-way interaction appears to be present, the results of the Kruskal-Wallis test should be viewed with caution. The only conclusion that can be stated is that the mean number of multinational force ships that

survive the two-day battles increases when the U. S. forces utilize both satellite intelligence and an additional carrier.

Cube Plot - Means for MOE 4
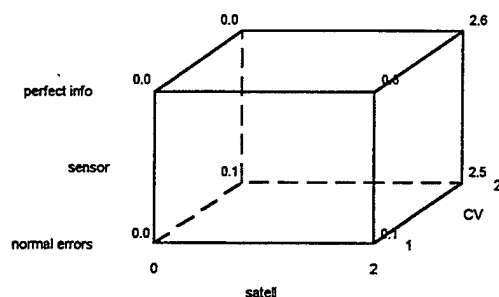
satellites	sensor accuracy	aircraft carriers	mean	se mean	std dev
0	normal errors	1	0	0.00	0.00
2	normal errors	1	0.1	0.10	0.32
0	perfect information	1	0	0.00	0.00
2	perfect information	1	0.6	0.22	0.70
0	normal errors	2	0.1	0.10	0.32
2	normal errors	2	2.5	0.31	0.97
0	perfect information	2	0	0.00	0.00
2	perfect information	2	2.6	0.31	0.97

Figure 11. Means of the Number of Multinational Force Ships Remaining at the Conclusion of the Two-Day Battle and Summary Statistics.

7. MOE 5

The data points for MOE 5 appear normally distributed as shown in the plot included in Appendix F. A factorial fit was computed and a plot of the residuals was generated to check the validity of the fit. The residual plot, which is displayed in Figure 12, refutes the normality assumption and the factorial fit was discarded. The Kruskal-Wallis tests, which are summarized in Table 7, once again indicate that

Table 7. Kruskal-Wallis Analysis for MOE 5

factor	H-statistic	p-value
satellite	9.25	0.00
sensor	0.01	0.91
carrier	44.88	0.00

Normal Probability Plot of the Residuals
(response is moe 5)
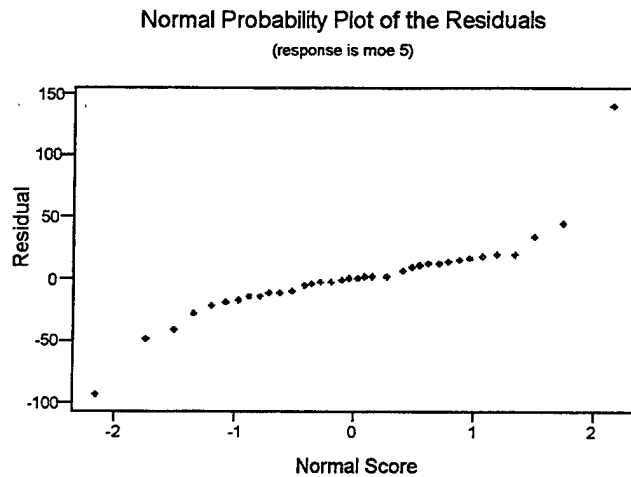
Figure 12. Normal Probability Plot of the Residuals for the Fitted Factorial Model of MOE 5

both the addition of satellite intelligence and a second carrier are significant. It can be seen in Figure 13 that when a second carrier is added, the mean number of U. S. missiles and attack aircraft launched against Iranian targets significantly increases. The mean number of missiles and attack aircraft launched also increases whenever satellite information is available, but in the case when only one carrier is in the battle group; this perceived increase is within the bounds of the standard errors of the means. This leads one to believe that the second carrier, which brings an additional air wing to both attack the Iranian targets, and provides a second set of reconnaissance aircraft, is the reason the mean number of missile and attack aircraft launches increases.

Cube Plot - Means for MOE 5
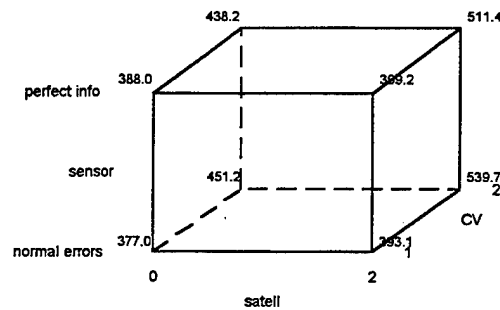
satellites	sensor accuracy	aircraft carriers	mean	se mean	std dev
0	normal errors	1	377	4.75	15.03
2	normal errors	1	393.1	6.09	19.25
0	perfect information	1	388	10.3	32.6
2	perfect information	1	399.2	5.77	18.26
0	normal errors	2	451.2	22.6	71.5
2	normal errors	2	539.7	8.28	26.19
0	perfect information	2	438.2	9.83	31.07
2	perfect information	2	511.4	8.92	28.2

Figure 13. Means of the Combined Number of U. S. Missiles and Attack Aircraft Launched Against Iranian Targets over the Two-Day Battle and Summary Statistics.

8. MOE 6

Since the data points for MOE 6 appear normally distributed, as shown in the plot included in Appendix F, an ANOVA table was computed. The results of the ANOVA appear in Table 8 and indicate that there is at least one strong two-way interaction, with a p-value of .00. Due to the strong two-way interaction between an additional carrier and satellite intelligence (p-value =.00), the two must be considered jointly and no conclusions about the large main effects of the availability of satellite intelligence or adding an additional carrier can be stated based on this analysis technique. A normal probability plot of the residuals shown in Figure 14 supports the normality assumption which is required for this analysis.

Normal Probability Plot of the Residuals
(response is moe 6)
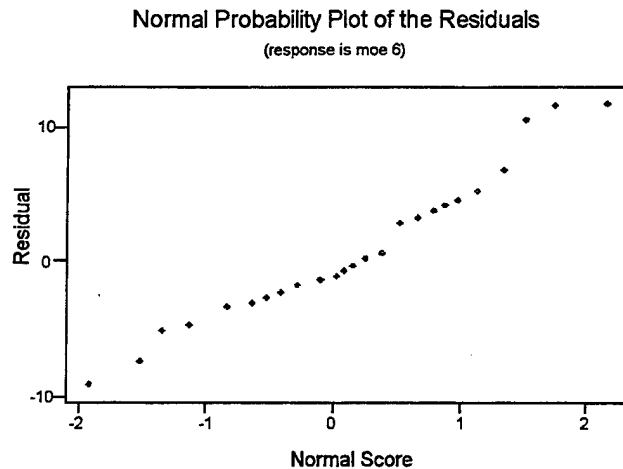
**Figure 14. Normal Probability Plot of the Residuals for the Fitted Factorial
Model of MOE 6**

The estimate of the two-way interaction of satellites and carriers suggests that the combined effect of using

satellite intelligence and an additional carrier suggests causes the mean number of U. S. missile and attack

aircraft launches to increase by about 7.

Table 8. Analysis of Variance Table for MOE 6 with Estimated Effects

Source	df	SS	MS	F-ratio	p-value
Main effects	3	8120.90	2706.98	63.55	0.00
Two-way interactions	3	2167.20	722.41	16.96	0.00
Three-way interaction	1	56.11	56.11	1.32	0.26
Residual error	72	3067.10	42.60		
Total	79	13411.40			

Factor	Effect	p-value
mean	104.79	-
satellite	17.13	0.00
sensor	2.03	0.17
carrier	10.43	0.00
satellite & sensor	0.58	0.70
satellite & carrier	10.38	0.00
sensor & carrier	-0.63	0.67
all	-1.68	0.26

Note: The value for the mean effect is the overall mean value of MOE 6 for the eighty
simulation runs. The values for the other effects represent the estimated change in MOE 6 when an
individual factor or combination of factors are at their high level (+) as compared to the when all of the
factors are at their low levels (-).The p-value of the estimated effect is the probability that the value for the
effect is actually zero.

Further examination of the cube plot of the means in Figure 13 shows that the same effect is occurring as in MOE 5. The addition of a carrier causes the mean number of hits scored against Iranian targets to increase as more attack aircraft are launched due to the additional airwing located on the second carrier. The increase when there is one carrier and satellite intelligence available falls within the standard deviations of the compared means. A definitive statement about the one way effect of satellite intelligence is questionable since it may be accounted for by the standard errors of the means.

Cube Plot - Means for MOE 6
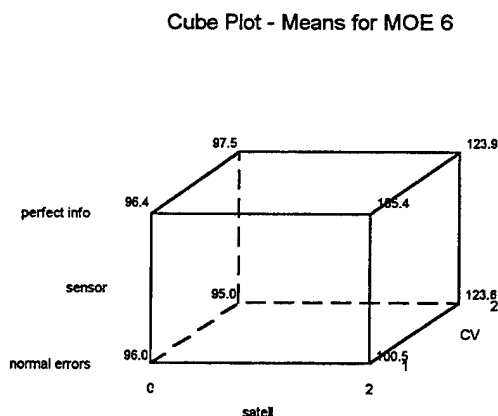
satellites	sensor accuracy	aircraft carriers	mean	se mean	std dev
0	normal errors	1	96	1.76	5.56
2	normal errors	1	100.5	1.80	5.70
0	perfect information	1	96.4	2.09	6.60
2	perfect information	1	105.4	2.31	7.29
0	normal errors	2	95	1.33	4.22
2	normal errors	2	123.6	2.52	7.97
0	perfect information	2	97.5	1.34	4.25
2	perfect information	2	123.9	2.85	9.01

Figure 15. Means of the Combined Number of U. S. Missiles and Attack Aircraft Launched Against Iranian Targets over the Two-Day Battle and Summary Statistics

9. MOE 7

Since the data points for MOE 7 appear normally distributed, as shown in the plot in Appendix F, a fractional factorial fit was computed. A residual plot, which is shown in Figure 16, verifies that the assumption of normality is reasonable. The results of the factorial analysis performed for MOE 7 are summarized in Table 9. Since there are no significant two-way and three-way interactions, the use of satellite intelligence (p-value = 0.00) and perfect sensor information (p-value = 0.00) are identified as

significant main effects. The estimate of the main effect of satellites indicates that use of satellite

intelligence causes the mean number of attack aircraft launched against U. S. and multinational force

ships to decrease by about 36. The estimate of the main effect of sensor accuracy indicates that when

perfect sensor information is used the mean number of attack aircraft launched decrease by about 19.

Normal Probability Plot of the Residuals
(response is moe 7)
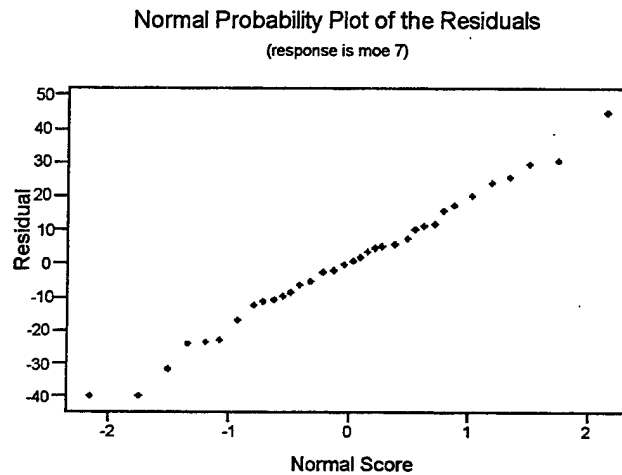
Figure 16. Normal Probability Plot of the Residuals for the Fitted Factorial Model of MOE 7

Table 9. Analysis of Variance Table for MOE 7 with Estimated Effects

Source	df	SS	MS	F-ratio	p-value
Main effects	3	32790.00	10930.00	20.17	0.00
Two-way interactions	3	4370.60	1456.90	2.69	0.05
Three-way interaction	1	583.20	583.20	1.08	0.30
Residual error	72	39008.00	541.80		
Total	79	76751.80			

Factor	Effect	p-value
mean	150.05	–
satellite	-35.90	0.00
sensor	-18.70	0.00
carrier	-1.00	0.85
satellite & sensor	9.10	0.09
satellite & carrier	-9.60	0.07
sensor & carrier	6.60	0.21
all	5.40	0.30

The Kruskal-Wallis tests and an examination of the cube plot in Figure 17 support these conclusions.

Table 10. Kruskal-Wallis Analysis for MOE 7

factor	H-statistic	p-value
satellite	27.62	0.00
sensor	6.40	0.01
carrier	.07	0.79

Cube Plot - Means for MOE 7
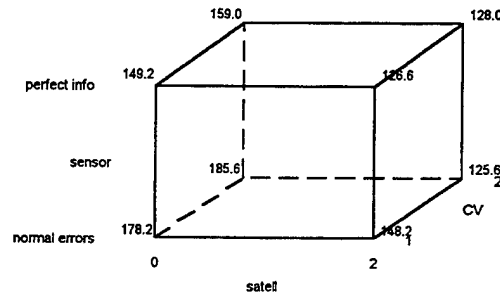
satellites	sensor accuracy	aircraft carriers	mean	se mean	std dev
0	normal errors	1	178.2	7.53	7.53
2	normal errors	1	148.2	8.76	27.69
0	perfect information	1	149.2	5.78	18.26
2	perfect information	1	126.6	9.32	29.49
0	normal errors	2	185.6	6.63	20.45
2	normal errors	2	125.6	7.17	22.66
0	perfect information	2	159	7.30	23.10
2	perfect information	2	128	5.58	17.64

Figure 17. Means of the Total Number of Iranian Attack Aircraft Launched Against U. S. ships over the Two-Day Battle and Summary Statistics

E. EXPERIMENT CONCLUSIONS

The track which the ships in the U. S. battle group follow provides protection by keeping them outside of the maximum weapon range of most Iranian weapons for significant portions of time over the two-day battle. Hence, MOE 1 and MOE 3 which measure the survivability of the ships among the CVBG do not show large responses to changes in each of the three conditions. The mean number of hits against these ships does decrease when perfect sensor information is used for a CVBG consisting of one carrier. The only other MOE which is affected by using perfect sensor information is MOE 7 which measures the number of attack aircraft sorties that

51

Iran launches at U. S. and multinational force ships. It appears that the use of perfect information by the U. S. increases the probability that a United States missile or aircraft strike scores a hit against an Iranian target, thereby reducing number of weapons that Iran can fire at the U. S. ships. It is concluded that the use of perfect sensor information increases the probability that ships among the carrier battle group survive over the two-day battle.

The track of the multinational force places these ships within Iranian weapons range early in the battle. Since reconnaissance aircraft are only flown from a carrier, the ability to locate and target Iranian SSM launchers and air bases which are located near the multinational force track does not occur until the CVBG nears the Straits of Hormuz. As a result, the specific scenario used in the simulation makes the destruction of all multinational force ships by Iran likely. The survivability of the multinational force is measured by MOE 2 and MOE 4. These measures show that the only condition that enables the some of the multinational force ships to successfully complete their transit occur when satellite intelligence is available and a second aircraft carrier is among the battle group. The availability of satellite intelligence provides the U. S. forces a method of detecting Iranian targets that are either not detected by shipboard sensors and reconnaissance aircraft, or are detected after they have launched a significant amount of weapons against the multinational force. Not only must the U. S. detect these targets, but additional strike assets in the form of the second carrier air wing must be available to attack these targets. Hence, for the scenario of the simulation, it is concluded that for the CVBG to successfully accomplish its mission, which is to ensure the multinational force reaches the Arabian Sea, satellite intelligence and a second aircraft carrier are required.

The mean number of U. S. missiles and attack aircraft launched at Iranian targets (MOE 5) and the mean number of hits scored against Iranian targets (MOE 6) both increase when an additional carrier is added to the battle group. This is not a surprising conclusion. It appears that the availability of satellite intelligence may have also caused these MOE to increase, but the analysis shows that the change due to this availability may be due to normal variation.

VI. CONCLUSIONS AND FURTHER STUDY

A. CONCLUSIONS

The analysis of the results obtained from the factorial experiment demonstrate that the outcome of this simulation model is responsive to the changes related to sensors and weapons which were varied during the experiment. The simulation results for this one specific conflict show that the use of intelligence information obtained from satellite assets combined with the assistance of a second aircraft carrier enables the CVBG to accomplish its mission. When these conditions are not met, all multinational force ships are usually destroyed. The use of sensors which provide ground truth position reports for all detections does not significantly improve the mission success of the CVBG which implies that the simulation is not as sensitive to changes in sensor accuracy as desired.

The simulation model approach presented shows promise of having informative predictive capabilities, but it presently does not appear to possess sufficient detail to properly measure all aspects of C^4ISR. A more detailed study of this model is needed to properly evaluate its present capability to assess the impact of changes in C^4ISR and improve on its shortcomings. After conducting this initial experiment and analysis, it is clear that for this simulation model to be a useful predictive tool to a decision maker, it needs to incorporate more realistic data fusion and decision models (rules) under which the battle is to be conducted.

B. FURTHER STUDY

This is an initial approach to a stochastic simulation model which utilizes very simplistic decision algorithms. The first step in improving this simulation model so that it is a more powerful predictive tool is to refine the decision modules that address targeting, detection, conflict adjudication, and sensor employment. The algorithms developed do not follow any formal U. S. or Iranian doctrine. This model can be adapted to incorporate more detailed decision algorithms using U. S. naval and Iranian tactics and doctrine.

Another area where further study is recommended is in the detection model. The sensors are given generic probabilities of detection which are not conditioned on a specific type of target. Also the influence of weather, jamming, and decoys are not addressed and when incorporated may improve the realism of the model.

53

The experimentation performed using this simulation model was very limited in scope. The response of this simulation can be further studied by changing other parameters, such as detection probabilities, sensor detection ranges, weapons loadout, or weapons range.

LIST OF REFERENCES

Box, George E. P., J. Stuart Hunter, and William G. Hunter, *Statistics for Experimenters, An Introduction to Design, Data Analysis, and Model Building*, John Wiley and Sons, Inc., 1978.

Deitel, H. M., and P. J. Deitel, *C++ How to Program*, Prentice Hall, 1994.

Devore, Jay L., *Probability and Statistics for Engineering and the Sciences, Fourth Edition*, Duxbury Press, 1995.

Gibbons, Jean Dickinson, *Nonparametric Methods for Quantitative Analysis*, American Sciences Press, Inc., 1985.

Gunston, Bill, consultant editor, *The Encyclopedia of World Air Power*, Crescent Books, 1980.

Minitab® Reference Manual Release 11 for Windows, Minitab, Inc., 1996.

Sharpe, Richard, editor, *Jane's Fighting Ships*, Janes Information Group, Inc., 1996.

APPENDIX A. GEOGRAPHIC DATA BASE

X Left	X Right	Y Upper	Y Lower	Land or Sea	Weather
0	20	0	20	1 (Land)	1
20	40	0	20	2 (Sea)	1
40	60	0	20	2	1
60	80	0	20	2	1
80	100	0	20	2	1
100	120	0	20	1	1
120	140	0	20	1	1
140	160	0	20	1	1
160	180	0	20	1	1
180	200	0	20	1	1
200	220	0	20	1	1
220	240	0	20	1	1
240	260	0	20	1	1
260	280	0	20	1	1
280	300	0	20	1	1
300	320	0	20	1	1
320	340	0	20	1	1
340	360	0	20	1	1
360	380	0	20	1	1
380	400	0	20	1	1
400	420	0	20	1	1
420	440	0	20	1	1
440	460	0	20	1	1
460	480	0	20	1	1
480	500	0	20	1	1
500	520	0	20	1	1
520	540	0	20	1	1
540	560	0	20	1	1
560	580	0	20	1	1
580	600	0	20	1	1
600	620	0	20	1	1
620	640	0	20	1	1
640	660	0	20	1	1
660	680	0	20	1	1
680	700	0	20	1	1
0	20	20	40	1	1
20	40	20	40	2	1
40	60	20	40	2	1
60	80	20	40	2	1
80	100	20	40	2	1
100	120	20	40	1	1
120	140	20	40	1	1
140	160	20	40	1	1
160	180	20	40	1	1
180	200	20	40	1	1
200	220	20	40	1	1
220	240	20	40	1	1
240	260	20	40	1	1
260	280	20	40	1	1

280	300	20	40	1	1
300	320	20	40	1	1
320	340	20	40	1	1
340	360	20	40	1	1
360	380	20	40	1	1
380	400	20	40	1	1
400	420	20	40	1	1
420	440	20	40	1	1
440	460	20	40	1	1
460	480	20	40	1	1
480	500	20	40	1	1
500	520	20	40	1	1
520	540	20	40	1	1
540	560	20	40	1	1
560	580	20	40	1	1
580	600	20	40	1	1
600	620	20	40	1	1
620	640	20	40	1	1
640	660	20	40	1	1
660	680	20	40	1	1
680	700	20	40	1	1
0	20	40	60	1	1
20	40	40	60	2	1
40	60	40	60	2	1
60	80	40	60	2	1
80	100	40	60	2	1
100	120	40	60	2	1
120	140	40	60	1	1
140	160	40	60	1	1
160	180	40	60	1	1
180	200	40	60	1	1
200	220	40	60	1	1
220	240	40	60	1	1
240	260	40	60	1	1
260	280	40	60	1	1
280	300	40	60	1	1
300	320	40	60	1	1
320	340	40	60	1	1
340	360	40	60	1	1
360	380	40	60	1	1
380	400	40	60	1	1
400	420	40	60	1	1
420	440	40	60	1	1
440	460	40	60	1	1
460	480	40	60	1	1
480	500	40	60	1	1
500	520	40	60	1	1
520	540	40	60	1	1
540	560	40	60	1	1
560	580	40	60	1	1
580	600	40	60	1	1

600	620	40	60	1	1
620	640	40	60	1	1
640	660	40	60	1	1
660	680	40	60	1	1
680	700	40	60	1	1
0	20	60	80	1	1
20	40	60	80	2	1
40	60	60	80	2	1
60	80	60	80	2	1
80	100	60	80	2	1
100	12o	60	80	2	1
120	140	60	80	1	1
140	160	60	80	1	1
160	180	60	80	1	1
180	200	60	80	1	1
200	220	60	80	1	1
220	240	60	80	1	1
240	260	60	80	1	1
260	280	60	80	1	1
280	300	60	80	1	1
300	320	60	80	1	1
320	340	60	80	1	1
340	360	60	80	1	1
360	380	60	80	1	1
380	400	60	80	1	1
400	420	60	80	1	1
420	440	60	80	1	1
440	460	60	80	1	1
460	480	60	80	1	1
480	500	60	80	1	1
500	520	60	80	1	1
520	540	60	80	1	1
540	560	60	80	1	1
560	580	60	80	1	1
580	600	60	80	1	1
600	620	60	80	1	1
620	640	60	80	1	1
640	660	60	80	1	1
660	680	60	80	1	1
680	700	60	80	1	1
0	20	80	100	1	1
20	40	80	100	1	1
40	60	80	100	2	1
60	80	80	100	2	1
80	100	80	100	2	1
100	120	80	100	2	1
120	140	80	100	2	1
140	160	80	100	1	1
160	180	80	100	1	1
180	200	80	100	1	1
200	220	80	100	1	1

220	240	80	100	1	1
240	260	80	100	1	1
260	280	80	100	1	1
280	300	80	100	1	1
300	320	80	100	1	1
320	340	80	100	1	1
340	360	80	100	1	1
360	380	80	100	1	1
380	400	80	100	1	1
400	420	80.	100	1	1
420	440	80	100	1	1
440	460	80	100	1	1
460	480	80	100	1	1
480	500	80	100	1	i
500	520	80	100	1	1
520	540	80	100	i	1
540	560	80	100	1	1
560	580	80	100	1	1
580	600	80	100	1	1
600	620	80	100	1	1
620	640	80	100	1	1
640	660	80	100	1	1
660	680	80	100	1	1
680	700	80	100	1	1
0	20	100	120	1	1
20	40	100	120	1	1
40	60	100	120	2	1
60	80	100	120	2	1
80	100	100	120	2	1
100	120	100	120	2	1
120	140	100	120	2	1
140	160	100	120	1	1
160	180	100	120	1	1
180	200	100	120	1	1
200	220	100	120	1	1
220	240	100	120	1	1
240	260	100	120	1	1
260	280	100	120	1	1
280	300	100	120	1	1
300	320	100	120	1	1
320	340	100	120	1	1
340	360	100	120	1	1
360	380	100	120	1	1
380	400	100	120	1	1
400	420	100	120	1	1
420	440	100	120	1	1
440	460	100	120	1	1
460	480	100	120	1	1
480	500	100	120	1	1
500	520	100	120	1	1
520	540	100	120	1	1

540	560	100	120	1	1
560	580	100	120	1	1
580	600	100	120	1	1
600	620	100	120	1	1
620	640	100	120	1	1
640	660	100	120	1	1
660	680	100	120	1	1
680	700	100	120	1	1
0	20	120	140	1	1
20	40	120	140	1	1
40	60	120	140	2	1
60	80	120	140	2	1
80	100	120	140	2	1
100	120	120	140	2	1
120	140	120	140	2	1
140	160	120	140	1	1
160	180	120	140	1	1
180	200	120	140	1	1
200	220	120	140	1	1
220	240	120	140	1	1
240	260	120	140	1	1
260	280	120	140	1	1
280	300	120	140	1	1
300	320	120	140	1	1
320	340	120	140	1	1
340	360	120	140	1	1
360	380	120	140	1	1
380	400	120	140	1	1
400	420	120	140	1	1
420	440	120	140	1	1
440	460	120	140	1	1
460	480	120	140	1	1
480	500	120	140	1	1
500	520	120	140	1	1
520	540	120	140	1	1
540	560	120	140	1	1
560	580	120	140	1	1
580	600	120	140	1	1
600	620	120	140	1	1
620	640	120	140	1	1
640	660	120	140	1	1
660	680	120	140	1	1
680	700	120	140	1	1
0	20	140	160	1	1
20	40	140	160	1	1
40	60	140	160	1	1
60	80	140	160	1	1
80	100	140	160	2	1
100	120	140	160	2	1
120	140	140	160	2	1
140	160	140	160	2	1

160	180	140	160	2	1
180	200	140	160	2	1
200	220	140	160	2	1
220	240	140	160	1	1
240	260	140	160	1	1
260	280	140	160	1	1
280	300	140	160	1	1
300	320	140	160	1	1
320	340	140	160	1	1
340	360	140	160	1	1
360	380	140	160	1	1
380	400	140	160	1	1
400	420	140	160	1	1
420	440	140	160	1	1
440	460	140	160	1	1
460	480	140	160	1	1
480	500	140	160	1	1
500	520	140	160	1	1
520	540	140	160	1	1
540	560	140	160	1	1
560	580	140	160	1	1
580	600	140	160	1	1
600	620	140	160	1	1
620	640	140	160	1	1
640	660	140	160	1	1
660	680	140	160	1	1
680	700	140	160	1	1
0	20	160	180	1	1
20	40	160	180	1	1
40	60	160	180	1	1
60	80	160	180	1	1
80	100	160	180	2	1
100	120	160	180	2	1
120	140	160	180	2	1
140	160	160	180	2	1
160	180	160	180	2	1
180	200	160	180	2	1
200	220	160	180	2	1
220	240	160	180	2	1
240	260	160	180	1	1
260	280	160	180	1	1
280	300	160	180	1	1
300	320	160	180	1	1
320	340	160	180	1	1
340	360	160	180	1	1
360	380	160	180	1	1
380	400	160	180	1	1
400	420	160	180	1	1
420	440	160	180	1	1
440	460	160	180	1	1
460	480	160	180	1	1

480	500	160	180	1	1
500	520	160	180	1	1
520	540	160	180	1	1
540	560	160	180	1	1
560	580	160	180	1	1
580	600	160	180	1	1
600	620	160	180	1	1
620	640	160	180	1	1
640	660	160	180	1	1
660	680	160	180	1	1
680	700	160	180	1	1
0	20	180	200	1	1
20	40	180	200	1	1
40	60	180	200	1	1
60	80	180	200	1	1
80	100	180	200	2	1
100	120	180	200	2	1
120	140	180	200	2	1
140	160	180	200	2	1
160	180	180	200	2	1
180	200	180	200	2	1
200	220	180	200	2	1
220	240	180	200	2	1
240	260	180	200	2	1
260	280	180	200	1	1
280	300	180	200	1	1
300	320	180	200	1	1
320	340	180	200	1	1
340	360	180	200	1	1
360	380	180	200	1	1
380	400	180	200	1	1
400	420	180	200	1	1
420	440	180	200	1	1
440	460	180	200	1	1
460	480	180	200	1	1
480	500	180	200	1	1
500	520	180	200	1	1
520	540	180	200	1	1
540	560	180	200	1	1
560	580	180	200	1	1
580	600	180	200	1	1
600	620	180	200	1	1
620	640	180	200	1	1
640	660	180	200	1	1
660	680	180	200	1	1
680	700	180	200	1	1
0	20	200	220	1	1
20	40	200	220	1	1
40	60	200	220	1	1
60	80	200	220	1	1
80	100	200	220	1	1

100	120	200	220	2	1
120	140	200	220	2	1
140	160	200	220	2	1
160	180	200	220	2	1
180	200	200	220	2	1
200	220	200	220	2	1
220	240	200	220	2	1
240	260	200	220	1	1
260	280	200	220	1	1
280	300	200	220	1	1
300	320	200	220	1	1
320	340	200	220	1	1
340	360	200	220	1	1
360	380	200	220	1	1
380	400	200	220	1	1
400	420	200	220	2	1
420	440	200	220	2	1
440	460	200	220	1	1
460	480	200	220	1	1
480	500	200	220	1	1
500	520	200	220	1	1
520	540	200	220	1	1
540	560	200	220	1	1
560	580	200	220	1	1
580	600	200	220	1	1
600	620	200	220	1	1
620	640	200	220	1	1
640	660	200	220	1	1
660	680	200	220	1	1
680	700	200	220	1	1
0	20	220	240	1	1
20	40	220	240	1	1
40	60	220	240	1	1
60	80	220	240	1	1
80	100	220	240	1	1
100	120	220	240	2	1
120	140	220	240	2	1
140	160	220	240	2	1
160	180	220	240	2	1
180	200	220	240	2	1
200	220	220	240	2	1
220	240	220	240	2	1
240	260	220	240	2	1
260	280	220	240	2	1
280	300	220	240	2	1
300	320	220	240	2	1
320	340	220	240	2	1
340	360	220	240	2	1
360	380	220	240	2	1
380	400	220	240	2	1
400	420	220	240	2	1

420	440	220	240	2	1
440	460	220	240	1	1
460	480	220	240	1	1
480	500	220	240	1	1
500	520	220	240	1	1
520	540	220	240	1	1
540	560	220	240	1	1
560	580	220	240	1	1
580	600	220	240	1	1
600	620	220	240	1	1
620	640	220	240	1	1
640	660	220	240	1	1
660	680	220	240	1	1
680	700	220	240	1	1
0	20	240	260	1	1
20	40	240	260	1	1
40	60	240	260	1	1
60	80	240	260	1	1
80	100	240	260	1	1
100	120	240	260	2	1
120	140	240	260	1	1
140	160	240	260	1	1
160	180	240	260	1	1
180	200	240	260	2	1
200	220	240	260	2	1
220	240	240	260	2	1
240	260	240	260	2	1
260	280	240	260	2	1
280	300	240	260	2	1
300	320	240	260	2	1
320	340	240	260	2	1
340	360	240	260	2	1
360	380	240	260	2	1
380	400	240	260	2	1
400	420	240	260	1	1
420	440	240	260	2	1
440	460	240	260	2	1
460	480	240	260	1	1
480	500	240	260	1	1
500	520	240	260	1	1
520	540	240	260	1	1
540	560	240	260	1	1
560	580	240	260	1	1
580	600	240	260	1	1
600	620	240	260	1	1
620	640	240	260	1	1
640	660	240	260	1	1
660	680	240	260	1	1
680	700	240	260	1	1
0	20	260	280	1	1
20	40	260	280	1	1

40	60	260	280	1	1
60	80	260	280	1	1
80	100	260	280	1	1
100	120	260	280	2	1
120	140	260	280	1	1
140	160	260	280	1	1
160	180	260	280	1	1
180	200	260	280	2	1
200	220	260	280	2	1
220	240	260	280	2	1
240	260	260	280	2	1
260	280	260	280	2	1
280	300	260	280	2	1
300	320	260	280	2	1
320	340	260	280	2	1
340	360	260	280	2	1
360	380	260	280	2	1
380	400	260	280	1	1
400	420	260	280	1	1
420	440	260	280	2	1
440	460	260	280	2	1
460	480	260	280	2	1
480	500	260	280	1	1
500	520	260	280	1	1
520	540	260	280	1	1
540	560	260	280	1	1
560	580	260	280	1	1
580	600	260	280	1	1
600	620	260	280	1	1
620	640	260	280	1	1
640	660	260	280	1	1
660	680	260	280	1	1
680	700	260	280	1	1
0	20	280	300	1	1
20	40	280	300	1	1
40	60	280	300	1	1
60	80	280	300	1	1
80	100	280	300	1	1
100	120	280	300	1	1
120	140	280	300	1	1
140	160	280	300	1	1
160	180	280	300	2	1
180	200	280	300	2	1
200	220	280	300	2	1
220	240	280	300	2	1
240	260	280	300	2	1
260	280	280	300	2	1
280	300	280	300	2	1
300	320	280	300	2	1
320	340	280	300	2	1
340	360	280	300	2	1

360	380	280	300	1	1
380	400	280	300	1	1
400	420	280	300	1	1
420	440	280	300	2	1
440	460	280	300	2	1
460	480	280	300	2	1
480	500	280	300	2	1
500	520	280	300	2	1
520	540	280	300	2	1
540	560	280	300	1	1
560	580	280	300	1	1
580	600	280	300	1	1
600	620	280	300	1	1
620	640	280	300	1	1
640	660	280	300	1	1
660	680	280	300	1	1
680	700	280	300	1	1
0	20	300	320	1	1
20	40	300	320	1	1
40	60	300	320	1	1
60	80	300	320	1	1
80	100	300	320	1	1
100	120	300	320	1	1
120	140	300	320	1	1
140	160	300	320	1	1
160	180	300	320	1	1
180	200	300	320	2	1
200	220	300	320	2	1
220	240	300	320	2	1
240	260	300	320	2	1
260	280	300	320	2	1
280	300	300	320	2	1
300	320	300	320	2	1
320	340	300	320	2	1
340	360	300	320	2	1
360	380	300	320	1	1
380	400	300	320	1	1
400	420	300	320	1	1
420	440	300	320	2	1
440	460	300	320	2	1
460	480	300	320	2	1
480	500	300	320	2	1
500	520	300	320	2	1
520	540	300	320	2	1
540	560	300	320	2	1
560	580	300	320	2	1
580	600	300	320	2	1
600	620	300	320	2	1
620	640	300	320	2	1
640	660	300	320	2	1
660	680	300	320	2	1

680	700	300	320	2	1
0	20	320	340	1	1
20	40	320	340	1	1
40	60	320	340	1	1
60	80	320	340	1	1
80	100	320	340	1	1
100	120	320	340	1	1
120	140	320	340	1	1
140	160	320	340	1	1
160	180	320	340	1	1
180	200	320	340	2	1
200	220	320	340	2	1
220	240	320	340	2	1
240	260	320	340	2	1
260	280	320	340	2	1
280	300	320	340	2	1
300	320	320	340	2	1
320	340	320	340	2	1
340	360	320	340	1	1
360	380	320	340	1	1
380	400	320	340	1	1
400	420	320	340	1	1
420	440	320	340	2	1
440	460	320	340	2	1
460	480	320	340	2	1
480	500	320	340	2	1
500	520	320	340	2	1
520	540	320	340	2	1
540	560	320	340	2	1
560	580	320	340	2	1
580	600	320	340	2	1
600	620	320	340	2	1
620	640	320	340	2	1
640	660	320	340	2	1
660	680	320	340	2	1
680	700	320	340	2	1
0	20	340	360	1	1
20	40	340	360	1	1
40	60	340	360	1	1
60	80	340	360	1	1
80	100	340	360	1	1
100	120	340	360	1	1
120	140	340	360	1	1
140	160	340	360	1	1
160	180	340	360	1	1
180	200	340	360	2	1
200	220	340	360	2	1
220	240	340	360	2	1
240	260	340	360	2	1
260	280	340	360	2	1
280	300	340	360	2	1

300	320	340	360	2	1
320	340	340	360	2	1
340	360	340	360	1	1
360	380	340	360	1	1
380	400	340	360	1	1
400	420	340	360	1	1
420	440	340	360	1	1
440	460	340	360	2	1
460	480	340	360	2	1
480	500	340	360	2	1
500	520	340	360	2	1
520	540	340	360	2	1
540	560	340	360	2	1
560	580	340	360	2	1
580	600	340	360	2	1
600	620	340	360	2	1
620	640	340	360	2	1
640	660	340	360	2	1
660	680	340	360	2	1
680	700	340	360	2	1
0	20	360	380	1	1
20	40	360	380	1	1
40	60	360	380	1	1
60	80	360	380	1	1
80	100	360	380	1	1
100	120	360	380	1	1
120	140	360	380	1	1
140	160	360	380	1	1
160	180	360	380	1	1
180	200	360	380	2	1
200	220	360	380	2	1
220	240	360	380	2	1
240	260	360	380	2	1
260	280	360	380	2	1
280	300	360	380	2	1
300	320	360	380	2	1
320	340	360	380	2	1
340	360	360	380	1	1
360	380	360	380	1	1
380	400	360	380	1	1
400	420	360	380	1	1
420	440	360	380	1	1
440	460	360	380	2	1
460	480	360	380	2	1
480	500	360	380	2	1
500	520	360	380	2	1
520	540	360	380	2	1
540	560	360	380	2	1
560	580	360	380	2	1
580	600	360	380	2	1
600	620	360	380	2	1

620	640	360	380	2	1
640	660	360	380	2	1
660	680	360	380	2	1
680	700	360	380	2	1
0	20	380	400	1	1
20	40	380	400	1	1
40	60	380	400	1	1
60	80	380	400	1	1
80	100	380	400	1	1
100	120	380	400	1	1
120	140	380	400	1	1
140	160	380	400	1	1
160	180	380	400	1	1
180	200	380	400	2	1
200	220	380	400	2	1
220	240	380	400	2	1
240	260	380	400	2	1
260	280	380	400	2	1
280	300	380	400	2	1
300	320	380	400	2	1
320	340	380	400	2	1
340	360	380	400	1	1
360	380	380	400	1	1
380	400	380	400	1	1
400	420	380	400	1	1
420	440	380	400	1	1
440	460	380	400	1	1
460	480	380	400	2	1
480	500	380	400	2	1
500	520	380	400	2	1
520	540	380	400	2	1
540	560	380	400	2	1
560	580	380	400	2	1
580	600	380	400	2	1
600	620	380	400	2	1
620	640	380	400	2	1
640	660	380	400	2	1
660	680	380	400	2	1
680	700	380	400	2	1

APPENDIX B. FORCE STRUCTURE DATA BASE

Unit #	Description	Hits to Kill	Unit Type	Op Speed	Not Used	# Missiles or A/C	Eff. Range
1	U.S. CV	40	1	20	0	46	550
2	U.S. warship	10	2	20	0	40	600
3	U.S. warship	10	2	20	0	40	600
4	U.S. warship	10	2	20	0	40	600
5	U.S. warship	10	2	20	0	40	600
6	U.S. warship	10	2	20	0	40	600
7	AOR	8	3	20	0	0	0
8	U.S. warship	10	2	15	0	40	600
9	U.S. warship	10	2	15	0	40	600
10	merchant	8	3	15	0	0	0
11	merchant	8	3	15	0	0	0
12	E-2C	4	12	100	0	0	0
13	U.S. recon a/c	2	13	325	0	0	0
14	U.S. recon a/c	2	13	325	0	0	0
15	U.S. recon a/c	2	13	200	0	0	0
16	Satellite	400	19	150	0	0	0
17	Satellite	400	19	150	0	0	0
18	Satellite	400	19	50	0	0	0
19	Satellite	400	19	50	0	0	0
20	U.S. CV	40	1	20	0	46	550
21	Iranian air base	60	21	0	0	50	230
22	Iranian air base	60	21	0	0	50	230
23	Iranian radar	8	22	0	0	0	0
24	Iranian radar	8	22	0	0	0	0
25	Iranian radar	8	22	0	0	0	0
26	Iranian radar	8	22	0	0	0	0
27	Iranian SSM	1	24	15	0	16	50
28	Iranian SSM	1	24	15	0	16	50
29	Iranian SSM	1	24	15	0	16	50
30	Iranian SSM	1	24	15	0	16	50
31	Iranian SSM	1	24	15	0	16	50
32	Iranian SSM	1	24	15	0	16	50
33	Iranian SSM	1	24	15	0	16	50
34	Iranian SSM	1	24	15	0	16	50
35	Iranian SSM	1	24	15	0	16	50
36	Iranian SSM	1	24	15	0	16	50
37	Iranian SSM	1	24	15	0	16	50
38	Iranian recon a/c	6	33	600	0	0	0
39	Iranian recon a/c	6	33	600	0	0	0
40	Iranian recon a/c	6	33	600	0	0	0

Unit Type: 1 = aircraft carrier, 2 = U.S. warship, 3 = merchant, 12 = E2-C aircraft, 13 = U.S.reconaissance aircraft, 19 = U.S. satellite, 21 = Iranian air base, 22 = Iranian long range radar site, 24 = Iranian SSM launcher, 33 = Iranian reconnaissance aircraft

Description: Included in the Appendix for clarity, it is not in the actual input data file

Op Speed: Normal operating speed of the unit in nautical miles per hour for ships and aircraft, miles per hour for SSM launchers.

Missiles or A/C: Number of surface-to-land missiles on board a warship, surface-to-surface missiles for an SSM launcher, or attack aircraft on board a carrier or air base.

Eff. Range: The effective range of the missile or aircraft in miles, used as a maximum range.

Unit #	Description	# Fighter A/C	Eff. Range	Surface sens	Air sens	ESM sen	X(0)	Y(0)
1	U.S. CV	30	200	2	8	2	695	395
2	U.S. warship	0	0	2	8	2	695	393
3	U.S. warship	0	0	2	8	2	694	393
4	U.S. warship	0	0	2	8	2	696	393
5	U.S. warship	0	0	2	8	2	694	395
6	U.S. warship	0	0	2	8	2	696	395
7	AOR	0	0	2	3	0	695	397
8	U.S. warship	0	0	2	8	2	250	360
9	U.S. warship	0	0	2	8	2	252	362
10	merchant	0	0	2	3	0	249	362
11	merchant	0	0	2	3	0	250	362
12	E-2C	0	0	4	6	6	695	395
13	U.S. recon a/c	0	0	2	4	2	695	395
14	U.S. recon a/c	0	0	2	4	2	695	395
15	U.S. recon a/c	0	0	2	4	2	695	395
16	Satellite	0	0	4	6	4	699	200
17	Satellite	0	0	4	6	4	699	250
18	Satellite	0	0	4	6	4	400	399
19	Satellite	0	0	4	6	4	500	399
20	Satellite	30	200	2	4	2	695	396
21	Iranian air base	30	335	2	4	4	410	135
22	Iranian air base	50	335	4	4	4	610	250
23	Iranian radar	0	0	4	4	4	340	210
24	Iranian radar	0	0	4	4	4	405	175
25	Iranian radar	0	0	4	4	4	490	250
26	Iranian radar	0	0	4	4	4	650	295
27	Iranian SSM	0	0	2	4	2	290	200
28	Iranian SSM	0	0	2	4	2	325	205
29	Iranian SSM	0	0	2	4	2	380	205
30	Iranian SSM	0	0	2	4	2	390	202
31	Iranian SSM	0	0	2	4	2	410	195
32	Iranian SSM	0	0	2	4	2	425	195
33	Iranian SSM	0	0	2	4	2	450	215
34	Iranian SSM	0	0	2	4	2	450	230
35	Iranian SSM	0	0	2	4	2	470	250
36	Iranian SSM	0	0	2	4	2	485	265
37	Iranian SSM	0	0	2	4	2	500	270
38	Iranian recon a/c	0	0	2	4	2	610	250
39	Iranian recon a/c	0	0	2	4	2	610	250
40	Iranian recon a/c	0	0	2	4	2	410	135

Fighter A/C: Initial number of fighter aircraft allocated to a U.S. CV or an Iranian air base.
Eff. Range: The effective range of the fighter aircraft in miles, used as a maximum range.
Surface sens: Surface sensor range in grid units, 1 unit = 20 miles.
Air sens: Air sensor range in grid units, 1 unit = 20 miles.
ESM sens: Electronic supports measures range in grid units, 1 unit = 20 miles.
X(0), Y(0): Initial (time = 0) x and y positions of a unit in cartesian coordinates.

Unit #	Description	X(1)	Y(1)	Loiter(1)	X(2)	Y(2)	Loiter(2)	X(3)	Y(3)	Loiter(3)	X(4)	Y(4)	Loiter(4)
1	U.S. CV	460	395	0	430	330	720	460	395	720	595	395	120
2	U.S. warship	460	393	0	430	328	720	460	393	720	595	393	120
3	U.S. warship	459	393	0	429	328	720	459	393	720	594	393	120
4	U.S. warship	461	393	0	431	328	720	461	393	720	596	393	120
5	U.S. warship	459	395	0	429	330	720	459	395	720	594	395	120
6	U.S. warship	461	395	0	431	330	720	461	395	720	596	395	120
7	AOR	460	397	0	430	332	720	460	397	720	595	397	120
8	U.S. warship	330	375	0	330	315	0	397	237	0	422	237	0
9	U.S. warship	332	377	0	332	315	0	397	239	0	422	239	0
10	merchant	329	377	0	329	315	0	397	239	0	422	239	0
11	merchant	330	377	0	330	315	0	397	239	0	422	239	0
12	E-2C	630	350	0	500	350	0	695	395	0	695	395	10
13	U.S. recon a/c	680	295	0	640	295	0	695	395	0	695	395	15
14	U.S. recon a/c	640	295	0	590	295	0	695	395	0	695	395	20
15	U.S. recon a/c	590	295	0	540	295	0	695	395	0	695	395	25
16	Satellite	10	200	240	699	200	240	10	200	240	699	200	240
17	Satellite	10	250	120	699	250	240	10	250	240	699	250	240
18	Satellite	400	10	120	400	399	120	400	10	120	400	399	120
19	Satellite	500	10	240	500	399	120	500	10	120	500	399	120
20	Satellite	460	396	0	430	331	0	460	396	0	595	396	120
21	Iranian air base	410	135	9000	410	135	9000	410	135	9000	410	135	9000
22	Iranian air base	610	250	9000	610	250	9000	610	250	9000	610	250	9000
23	Iranian radar	340	210	9000	340	210	9000	340	210	9000	340	210	9000
24	Iranian radar	405	175	9000	405	175	9000	405	175	9000	405	175	9000
25	Iranian radar	490	250	9000	490	250	9000	490	250	9000	490	250	9000
26	Iranian radar	650	295	9000	650	295	9000	650	295	9000	650	295	9000
27	Iranian SSM	300	150	60	300	150	60	300	150	60	300	150	60
28	Iranian SSM	300	150	60	300	150	60	300	150	60	300	150	60
29	Iranian SSM	300	150	60	300	150	60	300	150	60	300	150	60
30	Iranian SSM	300	150	60	300	150	60	300	.150	60	300	150	60
31	Iranian SSM	300	150	60	300	150	60	300	150	60	300	150	60
32	Iranian SSM	300	150	60	300	150	60	300	150	60	300	150	60
33	Iranian SSM	550	200	60	550	200	60	550	200	60	550	200	60
34	Iranian SSM	550	200	60	550	200	60	550	200	60	550	200	60
35	Iranian SSM	550	200	60	550	200	60	550	200	60	550	200	60
36	Iranian SSM	550	200	60	550	200	60	550	200	60	550	200	60
37	Iranian SSM	550	200	60	550	200	60	550	200	60	550	200	60
38	Iranian recon a/c	665	345	0	565	345	0	610	250	0	610	250	30
39	Iranian recon a/c	505	385	0	555	385	0	610	250	0	610	250	15
40	Iranian recon a/c	475	295	0	430	295	0	410	135	0	410	135	30

X(1), Y(1): X and y coordinates of the first waypoint for all units exept stationary and SSM units. For stationary units (Iranian air base and radar sites) all waypoints are equal to X(0) and Y(0). For an SSM launcher this is the location of the reload site.

Loiter(1): Amount of time units remain at the first waypoint, prior to moving towards the 2nd waypoint. For stationary units this number is greater than the total time of the battle. For SSM units, this is the time it remains at the reload site replenishing its missile supply.

X(2), Y(2), X(3), Y(3), X(4), Y(4): X and y coordinates of the second, third, and fourth waypoints for all units exept stationary and SSM units. No meaning for stationary and SSM units.

Loiter(2), Loiter(3), Loiter(4): Amount of time units remain at the second, thrid, or fourth waypoints. No meaning for stationary and SSM units. For a reconnaissance aircraft, Loiter(4) it is the amount of time it remains at home base until commencing another reconnaissance flight.

Unit #	Description	X(5)	Y(5)	Loiter(5)	X(6)	Y(6)	Loiter(6)
1	U.S. CV	695	395	1440	699	399	9000
2	U.S. warship	695	393	1440	699	399	9000
3	U.S. warship	694	393	1440	699	399	9000
4	U.S. warship	696	393	1440	699	399	9000
5	U.S. warship	694	395	1440	699	399	9000
6	U.S. warship	696	395	1440	699	399	9000
7	AOR	695	397	1440	699	399	9000
8	U.S. warship	460	370	0	690	399	9000
9	U.S. warship	460	370	0	690	399	9000
10	merchant	460	370	0	690	399	9000
11	merchant	460	370	0	690	399	9000
12	E-2C	55	45	1	130	45	1
13	U.S. recon a/c	15	100	1	40	100	1
14	U.S. recon a/c	55	100	1	50	100	1
15	U.S. recon a/c	105	100	1	50	100	9000
16	Satellite	10	200	240	699	200	9000
17	Satellite	10	250	240	699	250	9000
18	Satellite	400	10	120	400	399	9000
19	Satellite	500	10	120	500	399	9000
20	Satellite	695	396	1440	699	398	9000
21	Iranian air base	410	135	9000	410	135	9000
22	Iranian air base	610	250	9000	610	250	9000
23	Iranian radar	340	210	9000	340	210	9000
24	Iranian radar	405	175	9000	405	175	9000
25	Iranian radar	490	250	9000	490	250	9000
26	Iranian radar	650	295	9000	650	295	9000
27	Iranian SSM	300	150	60	300	150	9000
28	Iranian SSM	300	150	60	300	150	9000
29	Iranian SSM	300	150	60	300	150	9000
30	Iranian SSM	300	150	60	300	150	9000
31	Iranian SSM	300	150	60	300	150	9000
32	Iranian SSM	300	150	60	300	150	9000
33	Iranian SSM	550	200	60	550	200	9000
34	Iranian SSM	550	200	60	550	200	9000
35	Iranian SSM	550	200	60	550	200	9000
36	Iranian SSM	550	200	60	550	200	9000
37	Iranian SSM	550	200	60	550	200	9000
38	Iranian recon a/c	55	95	22	100	95	1
39	Iranian recon a/c	105	135	22	50	135	-1
40	Iranian recon a/c	65	160	21	45	160	1

X(5), Y(5): X and y coordinates of naval units fifth waypoint. No meaning for stationary and SSM units. For reconnaissance aircraft, these are the x and y distances, between its current location and its x and y destination on the first leg of a reconnaissance flight.

Loiter(5): Time naval units remain at the fifth waypoint. No meaning for stationary and SSM units. For reconnaissance aircraft, this is the Unit# of the air base or aircraft carrier that it is attached.

X(6), Y(6): X and y coordinates of naval units sixth waypoint. No meaning for stationary and SSM units. For reconnaissance aircraft, these are the x and y distances, between its first leg endpoint and and its x and y destination on the second leg of a reconnaissance flight.

Loiter(6): Amount of time naval units will remain at the sixth waypoint. For stationary, SSM, and U.S. reconnaissance aircraft units it has no meaning. For Iranian reconnaissance aircraft, this identifies the

direction from which the aircraft will fly away from the air base during its first leg of a reconnaissance flight.

APPENDIX C. ALGORITHMS

1. NAVAL SHIP MOVEMENT

```
IS THE SHIP IN MOTION?
        NO > LOITER TIME COMPLETE?
                NO > EXIT algorithm
                YES> Compute course & speed to next waypoint
                        Determine time to reach next waypoint
                                {Time = distance between waypoints/speed}
                        Place ship in motion
                        EXIT algorithm
        YES>  IS UNIT AT WAYPOINT?
                YES> WILL SHIP LOITER AT THIS POINT?
                        YES> Stop ship motion
                                Compute time at which ship will start movement
                NO>     Compute course & speed to next waypoint
                        Determine time to reach next waypoint
                        Place ship in motion
                EXIT algorithm
```

2. MOBILE SSM LAUNCHER MOVEMENT

```
IS THE CURRENT WAYPOINT THE RELOAD SITE?
        YES> IS THE SSM IN TRANSIT?
                YES>HAS IT REACHED THE SUPPLY DEPOT THIS TIME STEP?
                        YES > Stop SSM motion
                                Replenish missile supply
                                Compute time at which SSM will start movement
                                        {Time = current time + loiter time}
                        NO> Update SSM location
                        EXIT algorithm
                NO>LOITER TIME COMPLETE?
                        NO > EXIT algorithm
                        YES> Randomly compute next Waypoint,
                                Compute course & speed to next waypoint
                                Determine time to reach next waypoint
                                Place SSM in motion
                                EXIT algorithm
        NO> DOES THE LAUNCHER HAVE MISSILES?
                NO > Compute course & speed to reload site
                        Determine time to reach reload site
                        Place SSM in motion
                        Discontinue emitting
                        EXIT algorithm
                YES>  IS IT ON THE HOUR OR HALF HOUR?
                        IS U(0,1) < p_move?
                        NO > Use Bernoulli trial to determine if emitting
                                EXIT algorithm
                        YES>Randomly compute next waypoint
                                Discontinue emitting,
                        Compute course & speed to next waypoint
                                Determine time to reach next waypoint
                                Place SSM in motion
```

77

EXIT algorithm

3. RECONNAISSANCE AIRCRAFT MOVEMENT

IS THE CURRENT WAYPOINT NUMBER 1?
    YES> IS THE AIRCRAFT IN TRANSIT?
        NO> Compute course & speed to waypoint 1
            Determine time to reach waypoint 1
        YES> HAS THE A/C REACHED WAYPOINT 1?
            YES> Compute course & speed to waypoint 2
                Determine time to reach waypoint 2
                GOTO next step
            NO> GOTO next step
    Place aircraft in motion
    Commence emitting
    EXIT algorithm

    NO> IS THE CURRENT WAYPOINT NUMBER 2?
        YES> HAS THE AIRCRAFT REACHED THIS WAYPOINT?
            YES> Compute course & speed to waypoint 3
                Determine time to reach waypoint 3
                GOTO next step
            NO> GOTO next step
    Place aircraft in motion
    EXIT algorithm

    NO> IS THE CURRENT WAYPOINT NUMBER 3?
        YES> HAS THE AIRCRAFT REACHED THIS WP?
            YES> Update its x, y location to the CV or air base
                Stop aircraft motion
                Determine time at which next flight commences
                EXIT algorithm
            NO> Compute course & speed to CV or air base
                Determine time to reach home base
                Place aircraft in motion
                EXIT algorithm
      NO> IS THE A/C LOITERING AT ITS HOME BASE?
        YES> IS LOITER TIME COMPLETE?
            YES> Compute new waypoint 1
            NO> GOTO next step
         Update its x, y location to CV or air base
         EXIT algorithm
        NO> EXIT algorithm

4. ENGAGEMENT LIST (threat determination and closest unit)

Cycle through all U.S. contacts listed on the Iranian intelligence estimate
Beg
    Determine the type of U.S. unit
    Cycle through all Iranian units to determine which Iranian units are threatened
by the U.S. contact, and which Iranian unit will engage the U.S. contact
    Beg
        Determine distance between U.S. & Iranian unit
        IS DISTANCE < EFFECTIVE WEPS RANGE OF U.S. UNIT?

YES> Classify as high threat
NO> Classify as low threat
IF UNITS ARE AIR BASE AND NAVAL SHIP?
Beg
IS DISTANCE < CURRENT MIN AIR DISTANCE?
YES> Assigned attack unit = unit
Min air distance = distance
End
IF UNITS ARE SSM AND NAVAL SHIP?
Beg
IS DISTANCE < CURRENT MIN MISSILE DISTANCE?
YES> Assigned missile unit = unit
Min missile distance = distance
End
IF UNITS ARE AIR BASE AND U.S. RECON PLANE?
Beg
IS DISTANCE < CURRENT MIN AIR DISTANCE?
YES> Assigned air unit = unit
Min air distance = distance
End
WAS A MISSILE UNIT ASSIGNED?
YES> GOTO weapon selection algorithm
NO> WAS AN AIR UNIT ASSIGNED?
YES> GOTO weapon selection algorithm
NO> GOTO beginning of algorithm
WAS A WEAPON ASSIGNED TO STRIKE THE ENEMY?
YES> Compute conflict adjudication time
NO > GOTO beginning of algorithm

End

End

Cycle through all Iranian contacts listed on the U.S. intelligence estimate
Beg
Determine the type of Iranian unit
Cycle through all U.S. units to determine which U.S. units are threatened
by the Iranian contact, and which U.S. unit will engage the Iranian contact
Beg
Determine distance between Iranian & U.S. unit
IS DISTANCE < EFFECTIVE WEPS RANGE OF IRAN UNIT?
YES> Classify as high threat
NO> IS THE IRANIAN AIR BASE OR RADAR?
YES> Classify as medium threat
NO> Classify as high threat
IF UNITS ARE CV AND LAND BASED UNIT?
Beg
IS DISTANCE < CURRENT MIN AIR DISTANCE?
YES> Assigned attack unit = unit
Min air distance = distance
End
IF UNITS ARE WARSHIP AND LAND BASED UNIT?
Beg

79

IS DISTANCE < CURRENT MIN MISSILE DISTANCE?
YES> Assigned missile unit = unit
Min missile distance = distance
End
IF UNITS ARE CV AND IRANIAN RECON PLANE?
Beg
IS DISTANCE < CURRENT MIN AIR DISTANCE?
YES> Assigned air unit = unit
Min air distance = distance
End
GOTO weapon selection algorithm
WAS A WEAPON ASSIGNED TO STRIKE THE IRANIAN UNIT?
YES> Compute conflict adjudication time
NO > GOTO beginning of algorithm
End
End

5. IRANIAN WEAPON SELECTION AGAINST UNITED STATES UNITS

Determine the type of target
IS THE TARGET A RECONNAISSANCE AIRPLANE?
YES>   Identify the air base that is closest to the target and has fighter
aircraft available and within range of the target
NO>   1. Identify the SSM site that is closest to the target and has missiles
available and within range to strike the target
2. Identify the air base that is closest to the target and has attack
aircraft available that are within range to strike the target
ARE ANY OF THE IRANIAN UNITS WITHIN THE RANGE OF ANY WEAPON
CARRIED ONBOARD THE TARGET?
YES> Classify the target as a high threat
NO> Classify the target as a low threat

IF TARGET IS A/C CARRIER?
Beg
IF CLOSEST ASSET IS AN AIR BASE
IF THREAT IS HIGH
Assign Ha attack aircraft
ELSE
Assign La attack aircraft
IF CLOSEST ASSET IS SSM SITE
Launch all missiles left on launcher at A/C carrier
End
IF TARGET IS A SHIP
Beg
IF CLOSEST ASSET IS AN AIR BASE
IF THREAT IS HIGH
Assign Hs attack aircraft
ELSE
Assign Ls attack aircraft
IF CLOSEST ASSET IS AN SSM SITE
Launch Ms missiles at the ship
End
IF THE TARGET IS AN AIRPLANE

Beg
     Assign Hf fighters to engage the aircraft
End

6. UNITED STATES WEAPON SELECTION AGAINST IRANIAN UNITS

Determine the type of target
determine the friendly combat unit that is closest to the target
Classify the target as a high or low threat
IF THE TARGET IS AN AIR BASE
Beg
     IF THE THREAT IS HIGH
          Assign Ha attack aircraft
     ELSE
          Launch Ma cruise missiles at the target
End
IF THE TARGET IS A RADAR SITE
Beg
     IF THE THREAT IS HIGH
          Assign Hr attack aircraft
     ELSE
          Launch Mr cruise missiles at the target
End
IF THE TARGET IS A SSM SITE
Beg
     IF THE THREAT IS HIGH
          Assign Hs attack aircraft
     ELSE
          Launch Ms cruise missiles at the target
End
IF THE TARGET IS AN AIRPLANE
Beg
     Assign Hf fighters to engage the aircraft
End

7. CONFLICT ADJUDICATION

IS IT TIME TO ADJUDICATE THE CONFLICT?
     NO> exit algorithm
     YES> Complete the following cycle for total number of assets fired at target
          Beg. DID THE WEAPON EVADE THE TARGET'S DEFENSE
        SYSTEMS?
               NO> 1. Increment loss counter
                     2. Evaluate next asset or exit cycle
              YES> Did the weapon score a hit?
                     YES> Increment hit counter
          End
          Add the number of hits during this engagement to the targets
number of hits counter
          IS THE WEAPON A PLANE?
              NO> Go to next step
              YES> Return the number of aircraft that survived the attack to the
              number of assets available from the platform it was
launched
          IS THE TARGET AN AIR BASE OR AIRCRAFT CARRIER?

81

NO>Go to next step

YES>Compute the number of aircraft destroyed on the carrier or air base by the attack and remove them from the carrier or air base available assets

HAS THE NUMBER OF HITS TO KILL THE TARGET BEEN REACHED?

yes> 1. Remove the unit from the situation

2. Update the counter to indicate its destruction

no> Indicate that the unit is no longer targeted

APPENDIX D. ALGORITHM PARAMETERS

1. SENSOR DETECTION AND CONTACT LOCALIZATION

a. Probability of detection by a sensor

1. P{Surface search radar} = 0 .7
2. P{Air search radar} = 0.8
3. P{Electronic Support Measures (ESM)} = 0.6

b. Accuracy of sensor (σ)

1. Surface search radar: U.S. = 4 mile, Iranian = 4 mile
2. Air search radar: U.S. = .5 mile, Iranian = .5 mile
3. Electronic Support Measures: U.S. = 4 miles, Iranian = 4 miles

2. INTELLIGENCE UPDATE

If a unit has been detected, but has not been detected within the last T minutes, the unit is considered lost, and its position is considered unknown. T = 30 minutes.

3. WEAPON ALLOCATION :

a. Number of assets to allocate for a strike

1. 12 Iranian attack aircraft against a High threat carrier
2. 6 Iranian attack aircraft against a Low threat carrier
3. 4 Iranian attack aircraft against a High threat U.S. warship
4. 2 Iranian attack aircraft against a Low threat U.S. warship
5. 2 Surface-to-surface missiles against a U.S. warship
6. All remaining surface-to-surface missiles against a carrier at any threat level
7. 1 Fighter aircraft against a U.S. E-2C or reconnaissance airplane
8. 12 U.S. attack aircraft against a High threat air base
9. 4 U.S. attack aircraft against a Low threat air base
10. 6 Tomahawk missiles against a High threat air base
11. 6 Tomahawk missiles against a Low threat air base
12. 1 U.S. attack aircraft against a Medium threat Long range radar site
13. 4 Tomahawk missiles against a Medium threat Long range radar site
14. 6 U.S. attack aircraft against a Medium threat Long range radar site
15. 1 Tomahawk missile against a Medium threat Long range radar site
16. 1 attack aircraft against a SSM site at any threat level
17. 1 Tomahawk missile against a SSM site at any threat level
18. 1 U.S. fighter aircraft against a Iranian reconnaissance aircraft

b. Minimum weapons inventory (reserve on board)

1. 12 U.S. attack aircraft on board a carrier
2. 6 U.S. fighter aircraft on board a carrier
3. 20 Iranian attack aircraft at an air base
4. 8 Iranian fighter aircraft at an air base
5. 6 Tomahawk missiles on a U.S. warship

c. Speed of a weapon

1. 400 mph for a Tomahawk missile
2. 600 mph for a U.S. fighter aircraft
3. 400 mph for a U.S. attack aircraft
4. 500 mph for an Iranian surface-to-surface missile

5. 600 mph for an Iranian fighter aircraft
6. 400 mph for an Iranian attack aircraft

4. CONFLICT ADJUDICATION

a. Probability that a weapon is defeated by the target's defense mechanisms (P_{defeat})

1. P{U.S. attack aircraft against Iranian Long range radar site} = .3
2. P{U.S. attack aircraft against Iranian SSM site} = .1
3. P{U.S. attack aircraft against Iranian air base} = .3
4. P{U.S. fighter aircraft against Iranian reconnaissance aircraft} = .05
5. P{Tomahawk missile against Iranian long range radar site} = .3
6. P{Tomahawk missile against Iranian SSM site} = .2
7. P{Tomahawk missile against Iranian air base} = .3
8. P{Iranian attack aircraft against U.S. carrier} = .3
9. P{Iranian attack aircraft against U.S. warship} = .2
10. P{Iranian attack aircraft against U.S. merchant} = .2
11. P{Iranian fighter aircraft against U.S. E-2C or reconnaissance aircraft} = .05
12. P{Iranian surface-to-surface missile against U.S. carrier} = .1
13. P{Iranian surface-to-surface missile against U.S. warship} = .1
14. P{Iranian surface-to-surface missile against U.S. merchant} = .1

b. Probability a weapon scores a hit against a target given the weapon evaded the defense mechanisms of the target (P_{hit})

1. P{U.S. attack aircraft against Iranian Long range radar site} = .7
2. P{U.S. attack aircraft against Iranian SSM site} = .9
3. P{U.S. attack aircraft against Iranian air base} = .4
4. P{U.S. fighter aircraft against Iranian reconnaissance aircraft} = .1
5. P{Tomahawk missile against Iranian long range radar site} = .7
6. P{Tomahawk missile against Iranian SSM site} = .7
7. P{Tomahawk missile against Iranian air base} = .7
8. P{Iranian attack aircraft against U.S. carrier} = .8
9. P{Iranian attack aircraft against U.S. warship} = .8
10. P{Iranian attack aircraft against U.S. merchant} = .9
11. P{Iranian fighter aircraft against U.S. E-2C or reconnaissance aircraft} = .1
12. P{Iranian surface-to-surface missile against U.S. carrier} = .9
13. P{Iranian surface-to-surface missile against U.S. warship} = .9
14. P{Iranian surface-to-surface missile against U.S. merchant} = .9

c. Aircraft killed on the ground at an air base or carrier as a result of an enemy strike

1. Ac{U.S.} = .3
2. Ac{Iran} = .2

APPENDIX E. C++ COMPUTER SIMULATION CODE

```cpp
//  Edward R. Martinez
//  September 1996
//  MSDOS 6.2
//  Borland C++ 4.02 for windows
//  THIS IS FILE TerTest.cpp

#include <fstream.h>
#include <iostream.h>
#include <stdlib.h>
#include <math.h>
#include "terrain.h"

main(int argc, char *argv[])
{  int HalfHour;

// ALLOCATE MEMORY FOR TERRAIN MAP AND THEN READ INTO DATA STRUCTURE

    TerrainType myMap;
    ifstream infile(argv[1]);
    myMap.TerrainInput(infile);
    infile.close();
    myMap.Init_Units();

// OPEN FILES TO WRITE OUTPUT DATA

    ofstream outfile(argv[2]);
    ofstream display("Motion.dat");
    ofstream losses("Loss.dat");

//  VERIFY UNIT INITIAL CONDITIONS

    myMap.UnitPrint(outfile);

//  SIMULATION

    for (int clock = 1; clock <= 2880 ; clock++) {

        // IDENTIFY HALF HOUR  & OUTPUT TIME OF SIMULATION ON SCREEN

        HalfHour = (clock % 30 == 0)?1:0;
        if (clock == 1) {HalfHour = 1;}
        cout<<clock<<endl;

        //  BATTLE TIME STEP CYCLE (1 MINUTE TIME INTERVALS)

        myMap.Movement(HalfHour,clock);
        myMap.Sensors(clock,display);
        myMap.UpdateView(clock);
        myMap.Target(clock);
        myMap.Adjudicate(clock,losses);
        myMap.ReturnAircraft(clock);
    }
    //  WRITE RESULTS TO OUTPUT FILE

    myMap.Status(outfile,clock);
    myMap.WepsStatus(outfile);

    cout<<"End of Program"<<endl;
    return 1;
}
```

85

```cpp
//  Edward R. Martinez
//  September 1996
//  THESIS
//  MSDOS 6.22
//  Borland C++ 4.02 for windows
//  THIS IS FILE Terrain.h
//
//  This is the header file for the class TerrainType.  This class models a
//  region in the world.  It utilizes class Grid
//  The source file Terrain.cpp holds the code for the class.

#ifndef __Terrain_h
#define __Terrain_h

#define FAILOPEN 'F'
#define READDATA 'R'

#include <fstream.h>
#include <iostream.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>
#include "RdNumGen.h"
#include "Grid.h"
#include "Unit.h"
#include "Stack.h"

class TerrainType {

    public :

        TerrainType();
        ~TerrainType();
        char TerrainInput(ifstream&);
        void Init_Units();
        void TerrainPrint(ostream&);
        void UnitPrint(ostream&);
        void UnitDisplay(ostream&);
        void Movement(const int, const int);
        void Sensors(const int, ostream&);
        void UpdateView(const int);
        void Target(const int);
        void Adjudicate(const int, ostream&);
        void ReturnAircraft(const int);
        void Status(ostream&,int);
        void WepsStatus(ostream&);

    private :

        //  VARIABLES

        int NumberOfGrids, NumBlue, NumRed;
        GridType* TerrainMap[701];
        UnitType* ForceMap[41];
        RdGen *RandNumber;
        stackType *BlueTarget, *RedTarget, *RetAircraft;

        //  PARAMETERS AND COUNTERS

        double p_move,
                Pd_Bat_Rradar,Pd_Bat_RSSM, Pd_Bat_Rab,Pd_Bf_Rrecon,
                Pd_Bcm_Rradar,Pd_Bcm_RSSM,Pd_Bcm_Rab,
                Pd_Rat_Bcar,Pd_Rat_Bwar,Pd_Rat_Bmerch,
                Pd_Rf_Brecon,Pd_RSSM_Bcar, Pd_RSSM_Bwar,Pd_RSSM_Bmerch,
                Ph_Bat_Rradar,Ph_Bat_RSSM, Ph_Bat_Rab,Ph_Bf_Rrecon,
                Ph_Bcm_Rradar,Ph_Bcm_RSSM,Ph_Bcm_Rab,
```

```cpp
        Ph_Rat_Bcar, Ph_Rat_Bwar, Ph_Rat_Bmerch,
        Ph_Rf_Brecon, Ph_RSSM_Bcar, Ph_RSSM_Bwar, Ph_RSSM_Bmerch,
        IRMsl, USMsl, IRB, USB;

    int RHa, RLa, RHs, RLs, RMs, RHf,
        BHa, BLa, BMa, BHr, BMr, BHs, BMs, BHf,
        BlueCM, BlueCMLoss, BlueF, BlueA, IBlueF, IBlueA, BlueFLoss, BlueALoss,
        RedF, RedA, IRedF, IRedA, RedFLoss, RedALoss, RedSSM, RedSSMLoss,
        SP_Bcm, SP_Bf, SP_Bat, SP_Rcm, SP_Rf, SP_Rat,
        Batac, Bfac, Ratac, Rfac, Btom,
        ICarrier, IWarship, IMerchant, IBlue_Recon, IAirbase, ILong_Range_Radar,
        ISSM, IRed_Recon,
        ECarrier, EWarship, EMerchant, EBlue_Recon, EAirbase, ELong_Range_Radar,
        ESSM, ERed_Recon;

    // FUNCTIONS

    int maximum(const int a, const int b)  { return a>b ? a:b ;}
    int minimum(const int a, const int b)  { return a<b ? a:b ;}
    double minimum(const double a, const double b)  {return a<b ? a:b ;}
    int Getnumber(int Number)  {return ForceMap[Number]->GetNumber();}
    float GetXcoord(int Number)   {return ForceMap[Number]->GetxCoord();}
    float GetYcoord(int Number)   {return ForceMap[Number]->GetyCoord();}
    double DetDist(const int, const int);
    double DetDist(const int, const double, const double);
    void UnitInput();
    int DetermineGrid(const int);
    void StateGrid(int Number, int Grid){ForceMap[Number]->SpecifyGrid(Grid);}
    void SensorLook(const int, const int, const int, const int,
                                          const int, ostream&);
    int BlueWepSel(const int, int&, const int, const int, int&);
    int RedWepSel(const int, const int, const int, int&);
    int BStrike(const int, const int, const int, const int, const int, const double);
    int Fight(const double, const double, const int, int&);
    int RStrike(const int, const int, const int, const int, const int, const double);
    void UpdateCounter(const int);
    void InitCounter(const int, const int, const int, const int);
    void ACdamage(const int, const int, const int);

};

#endif
```

```
//   Edward R. Martinez
//   September 1996
//   THESIS
//   MSDOS 6.22
//   Borland C++ 4.02 for windows
//   THIS IS FILE Terrain.cpp
//
//   The class Terrain models the area of the World consisting of the Persian
//   Gulf, Gulf of Oman and Iran. It utilizes class GridType to hold individual
//   square miles of area.

#include <fstream.h>
#include <iostream.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>
#include "Terrain.h"
#include "RdNumGen.h"
#include "Grid.h"
#include "unit.h"


//   CONSTRUCTOR

TerrainType::TerrainType() {
    NumberOfGrids = NumBlue = NumRed = 0;
    BlueTarget = new stackType();
    RedTarget = new stackType();
    RetAircraft = new stackType();
    RandNumber = new RdGen();

    //   DEFINE CONSTANT VARIABLES

    p_move = 0.8; //   Probability a mobile SSM launcher will move every 1/2 hr

    //   Number of Assets to use in an attack

    RHa = 12; //   Red attack aircraft against High threat CV
    RLa =  6; //   Red attack aircraft against Low threat CV
    RHs =  4; //   Red attack aircraft against Hight threat warship
    RLs =  2;
    RMs =  2; //   Red SSM against U. S. targets
    RHf =  1;
    BHa = 6;  //   Blue attack aircraft against high threat air base
    BLa = 4;  //                                   low threat air base
    BMa = 6;  //   Blue cruise missile against air base
    BHr =  1; //   Blue cruise missile against high threat radar site
    BMr =  1; //                        against medium threat radar site
    BHs =  1; //
    BMs =  1; //
    BHf =  1;   //   Blue fighter against Red recon plane

    //   MINIMIM PERMISSIABLE LEVELS TO STRIKE
    Batac = 12;
    Bfac  = 10;
    Ratac = 20;
    Rfac  = 8;
    Btom  = 6;


    //   AIRCRAFT DAMAGED DURING A STRIKE ON CV OR AIR BASE

    IRMsl = .01;
    USMsl = .02;
    IRB   = .02;
    USB   = .02;

    //   COUNTER TO KEEP TRACK OF TOTAL FORCE WEAPONS USED/LOST
```
88

```
BlueCM      = 0;
BlueF       = 0;
BlueA       = 0;
IBlueF      = 0;
IBlueA      = 0;
BlueCMLoss  = 0;
BlueFLoss   = 0;
BlueALoss   = 0;
RedF        = 0;
RedA        = 0;
IRedF       = 0;
IRedA       = 0;
RedSSMLoss  = 0;
RedFLoss    = 0;
RedALoss    = 0;
RedSSM      = 0;

//   COUNTER TO KEEP TRACK OF UNITS LOST

ICarrier = 0;
IWarship = 0;
IMerchant = 0;
IBlue_Recon = 0;
IAirbase = 0;
ILong_Range_Radar = 0;
ISSM = 0;
IRed_Recon = 0;


ECarrier = 0;
EWarship = 0;
EMerchant = 0;
EBlue_Recon = 0;
EAirbase = 0;
ELong_Range_Radar = 0;
ESSM = 0;
ERed_Recon = 0;


SP_Bcm = 400;
SP_Bf  = 600;
SP_Bat = 400;
SP_Rcm = 500;
SP_Rf  = 600;
SP_Rat = 400;

//   PROBABILITY A STRIKE WEAPON IS DEFEATED BY TARGET DEFENSIVE MECHANISMS

Pd_Bat_Rradar   = .3;
Pd_Bat_RSSM     = .1;
Pd_Bat_Rab      = .3;
Pd_Bf_Rrecon    = .05;
Pd_Bcm_Rradar   = .3;
Pd_Bcm_RSSM     = .2;
Pd_Bcm_Rab      = .3;
Pd_Rat_Bcar     = .3;
Pd_Rat_Bwar     = .2;
Pd_Rat_Bmerch   = .2;
Pd_Rf_Brecon    = .05;
Pd_RSSM_Bcar    = .1;
Pd_RSSM_Bwar    = .1;
Pd_RSSM_Bmerch  = .1;
```

```
//   PROBABILTY A STRIKE WEAPON SCORES A DIRECT HIT AGAINST TARGET AFTER IT
//   AVOIDED DEFENSE MECHANISMS
```

```cpp
        Ph_Bat_Rradar  = .7;
        Ph_Bat_RSSM    = .9;
        Ph_Bat_Rab     = .4;
        Ph_Bf_Rrecon   = .1;
        Ph_Bcm_Rradar  = .7;
        Ph_Bcm_RSSM    = .7;
        Ph_Bcm_Rab     = .7;
        Ph_Rat_Bcar    = .8;
        Ph_Rat_Bwar    = .8;
        Ph_Rat_Bmerch  = .9;
        Ph_Rf_Brecon   = .1;
        Ph_RSSM_Bcar   = .9;
        Ph_RSSM_Bwar   = .9;
        Ph_RSSM_Bmerch = .9;

    for (int i = 0; i<= 700; i++) {
        TerrainMap[i] = new GridType();
    }
    for (int j = 0; j<= 40; j++)  {
        ForceMap[j] = new UnitType();

    }
}


// DESTRUCTOR

TerrainType::~TerrainType()    {
    NumberOfGrids = 0;
    delete BlueTarget;
    delete RedTarget;
    delete RetAircraft;
    delete RandNumber;

    for (int i = 0; i<= 700; i++) {
        delete TerrainMap[i];
    }
    for (int j = 0; j<= 40; j++)  {
        delete ForceMap[j];
    }
}

// INPUT THE TERRAIN AND INITIAL UNIT DATA BASES

char TerrainType::TerrainInput(ifstream& inputfile)    {

// DEFINE VARIABLES

    char *data, LineofData[160];
    int x1,x2,y1,y2,t,w,U,H,T,M,B,MS,MR,MSR,BR,SRR,ARR,ESMR,XP0,YP0,XP1,YP1;
    int LP1,XP2,YP2,LP2,XP3,YP3,LP3,XP4,YP4,LP4,XP5,YP5,LP5,XP6,YP6,LP6;

// READ IN NUMBER OF GRIDS FOR THE TERRAIN AND NUMBER OF UNITS FOR EACH FORCE

    inputfile.getline(LineofData,160,'\n');
    data = strtok(LineofData,",");
    NumberOfGrids = atoi(data);
    data = strtok(NULL,",");
    NumBlue = atoi(data);
    data = strtok(NULL,",");
    NumRed = atoi(data);

// READ IN X AND Y COORDINATES OF FOUR CORNERS OF TERRAIN GRID AND STORE AS
// A TERRAIN MAP

    for(int counter = 1; counter <= NumberOfGrids; counter++)     {
        inputfile.getline(LineofData,160,'\n');
```

```
        data = strtok(LineofData,",");
        x1 = atoi(data);
        data = strtok(NULL,",");
        x2 = atoi(data);
        data = strtok(NULL,",");
        y1 = atoi(data);
        data = strtok(NULL,",");
        y2 = atoi(data);
        data = strtok(NULL,",");
        t = atoi(data);
        data = strtok(NULL,",");
        w = atoi(data);

//      STORE AN INDIVIDUAL GRID

        TerrainMap[counter]->SetGridPoints(x1,x2,y1,y2,t,w);
    }


// READ IN THE UNIT INITIAL FORCE STRUCTURE AND STORE AS A UNIT MAP

    for(counter = 1; counter <= (NumBlue + NumRed); counter++)   {
        inputfile.getline(LineofData,160,'\n');
        data = strtok(LineofData,",");
        U = atoi(data);
        data = strtok(NULL,",");
        H = atoi(data);
        data = strtok(NULL,",");
        T = atoi(data);
        data = strtok(NULL,",");
        MS = atoi(data);
        data = strtok(NULL,",");
        MR = atoi(data);
        data = strtok(NULL,",");
        M = atoi(data);
        data = strtok(NULL,",");
        MSR = atoi(data);
        data = strtok(NULL,",");
        B = atoi(data);
        data = strtok(NULL,",");
        BR = atoi(data);
        data = strtok(NULL,",");
        SRR = atoi(data);
        data = strtok(NULL,",");
        ARR = atoi(data);
        data = strtok(NULL,",");
        ESMR = atoi(data);
        data = strtok(NULL,",");
        XP0 = atoi(data);
        data = strtok(NULL,",");
        YP0 = atoi(data);
        data = strtok(NULL,",");
        XP1 = atoi(data);
        data = strtok(NULL,",");
        YP1 = atoi(data);
        data = strtok(NULL,",");
        LP1 = atoi(data);
        data = strtok(NULL,",");
        XP2 = atoi(data);
        data = strtok(NULL,",");
        YP2 = atoi(data);
        data = strtok(NULL,",");
        LP2 = atoi(data);
        data = strtok(NULL,",");
        XP3 = atoi(data);
        data = strtok(NULL,",");
```

```
                YP3 = atoi(data);
                data = strtok(NULL,",");
                LP3 = atoi(data);
                data = strtok(NULL,",");
                XP4 = atoi(data);
                data = strtok(NULL,",");
                YP4 = atoi(data);
                data = strtok(NULL,",");
                LP4 = atoi(data);
                data = strtok(NULL,",");
                XP5 = atoi(data);
                data = strtok(NULL,",");
                YP5 = atoi(data);
                data = strtok(NULL,",");
                LP5 = atoi(data);
                data = strtok(NULL,",");
                XP6 = atoi(data);
                data = strtok(NULL,",");
                YP6 = atoi(data);
                data = strtok(NULL,",");
                LP6 = atoi(data);
                InitCounter(T,H,M,B);

                //  STORE AN INDIVIDUAL UNIT IN THE UNIT MAP

                ForceMap[counter]->SetUnit(U,H,T,SRR,ARR,ESMR,M,B,LP1,LP2,LP3,LP4,LP5,LP6,
                MS,MR,MSR,BR,XP0,YP0,XP1,YP1,XP2,YP2,XP3,YP3,XP4,YP4,XP5,YP5,XP6,YP6);
        }
        return READDATA;
}


//  READ IN INITIAL UNIT LOCATIONS TO THE GRID MAP

void TerrainType::Init_Units()    {
    int Gridnumber;
    for (int i = 1;i<=(NumBlue + NumRed);i++) {
        if (ForceMap[i]->GetType() != -1)    {
            Gridnumber = DetermineGrid(i);
            TerrainMap[Gridnumber]->AddUnit(i,1,ForceMap[i]->GetType(),
                ForceMap[i]->GetxCoord(),ForceMap[i]->GetyCoord());
            ForceMap[i]->SpecifyGrid(Gridnumber);
        }
    }
}


//  OUTPUT TerrainMap TO AN OUTPUT FILE

void TerrainType::TerrainPrint(ostream& method) {
    for(int i = 1; i <= NumberOfGrids; i++)    {
        if (TerrainMap[i]->AnyUnits())    {
            method<<"["<<i<<"] ";
            TerrainMap[i]->GridPrint(method);
            method<<endl;
        }
    }
}


//  OUTPUT UnitMap TO AN OUTPUT FILE

void TerrainType::UnitPrint(ostream &method) {
    for (int i = 1; i <= 40; i++) {ForceMap[i]->PrintUnit(method);}
    }


//  PRINT OUT X AND Y COORDINATES OF ALL UNITS TO AN OUTPUT FILE

void TerrainType::UnitDisplay(ostream &method)    {
```

92

```
        for (int i = 1; i <= 40; i++) {
            ForceMap[i]->PrintLoc(method);
        }
}


//   PERFORM MOVEMENT UPDATE STEP FOR ALL UNITS THAT ARE FUNCTIONING

void TerrainType::Movement(const int Flag, const int TheTime)  {
    int Update = 0, NewGrid, Unit,direct,AB;
    double cs;

    for (int i = 1; i<(NumBlue + NumRed + 1); i++)   {
        Unit = ForceMap[i]->GetType();
        if (Unit > 0)  {
            if (Unit < 11 || Unit == 19){Update = ForceMap[i]->MoveShip(TheTime);}
            else if (Unit < 20) {
                AB = ForceMap[i]->GetLP5();
                cs = ForceMap[AB]->GetCourse();
                cs = (cs > 0)?cs:-cs;
                direct = (cs >1.57)?-1:1;
                Update = ForceMap[i]->MoveAir(TheTime,ForceMap[AB]->GetxCoord(),
                                        ForceMap[AB]->GetyCoord(),direct,-1,0);
            }
            else if (Unit == 24) {
                Update = ForceMap[i]->MoveSSM(TheTime,Flag,p_move);
                if (Update == 1)   {
                    NewGrid = DetermineGrid(i);
                    if (TerrainMap[NewGrid]->GetType() == 1)   {
                        ForceMap[i]->ChangeSSM();
                    }
                }
            }
            else if (Unit > 30) {
                direct = ForceMap[i]->GetLP6();
                AB = ForceMap[i]->GetLP5();
                Update = ForceMap[i]->MoveAir(TheTime,ForceMap[AB]->GetxCoord(),
                                        ForceMap[AB]->GetyCoord(),direct,1,1);
            }
            else {Update = 0;}

            if (Update == 1)   {ForceMap[i]->UpdateMotion(.01667);}
            if (Update > 0)    {
                NewGrid = DetermineGrid(i);
                TerrainMap[ForceMap[i]->GetGrid()]->RemoveUnit(i);
                TerrainMap[NewGrid]->AddUnit(i,ForceMap[i]->GetEMCON(),
                ForceMap[i]->GetType(),ForceMap[i]->GetxCoord(),
                ForceMap[i]->GetyCoord());
                if (NewGrid != ForceMap[i]->GetGrid()) {
                    ForceMap[i]->SpecifyGrid(NewGrid);
                }
            }
        }
    }
}


//   LOOK AT THE WORLD WITH EACH OF THREE TYPES OF SENSORS

void TerrainType::Sensors(const int TheTime, ostream &method)   {
    int Gridnumber;
    int p = 1;
    method<<endl<<"Time = "<<TheTime<<endl;

    //   ALLOW EACH BLUE UNIT TO USE HIS THREE SENSORS TO LOOK FOR THE ENEMY

    for (p = 1; p < (NumBlue + 1); p++) {
        method<<p<<" ";
```

93

```cpp
            if (ForceMap[p]->GetType() > 0)    {
                Gridnumber = ForceMap[p]->GetGrid();
                method<<Gridnumber<<"  ";
                SensorLook(Gridnumber,0,ForceMap[p]->GetSRR(),TheTime,1,method);
                method<<"Surf ";
                SensorLook(Gridnumber,0,ForceMap[p]->GetARR(),TheTime,2,method);
                method<<"Air ";
                SensorLook(Gridnumber,0,ForceMap[p]->GetESMR(),TheTime,3,method);
                method<<"ESM "<<endl;
            }
    }


    //   ALLOW EACH RED UNIT TO USE HIS THREE SENSORS TO LOOK FOR THE ENEMY

    for (p = (NumBlue + 1); p < (NumBlue + NumRed + 1); p++)      {
        if (ForceMap[p]->GetType() > 0)    {
            method<<p<<"  ";
            Gridnumber = ForceMap[p]->GetGrid();
            method<<Gridnumber<<"  ";
            SensorLook(Gridnumber,1,ForceMap[p]->GetSRR(),TheTime,1,method);
            method<<"Surf ";
            SensorLook(Gridnumber,1,ForceMap[p]->GetARR(),TheTime,2,method);
            method<<"Air ";
            SensorLook(Gridnumber,1,ForceMap[p]->GetESMR(),TheTime,3,method);
            method<<"ESM "<<endl;
        }
    }
}


//   SEARCH FOR TARGETS WITH ALL A SENSOR

void TerrainType::SensorLook(const int GridNum, const int BlueRed,
    const int sensor_range,const int Timer, const int SensType, ostream &method){
    int row = 0;
    int col = 0;

    if (sensor_range  == 0)      {
        method<<"sr = "<<sensor_range;
        if (SensType == 1)    {TerrainMap[GridNum]->LookSurf(BlueRed,Timer,method);}
        else if (SensType == 2) {TerrainMap[GridNum]->LookAir(BlueRed,Timer,method);}
        else if (SensType == 3) {TerrainMap[GridNum]->LookESM(BlueRed,Timer,method);}
    }
    else   {
        method<<"srm = "<<sensor_range;
        row = GridNum/35;
        col = GridNum - row*35;
        if (col == 0) {row--; col = 35;}
        int col_low = maximum(1,col-sensor_range);
        int col_high = minimum(35,col+sensor_range);
        int row_low = maximum(0,row-sensor_range);
        int row_high = minimum(19,row+sensor_range);
        int deltacol = col_high - col_low;
        int deltarow = row_high - row_low;
        int start_point = 35*row_low + col_low;
        for (int r = 0; r <= deltarow; r++) {
            for (int c = 0; c <= deltacol; c++) {
                if (SensType == 1)
                    {TerrainMap[(start_point+35*r + c)]->LookSurf(BlueRed,Timer,method);}
                else if (SensType == 2)
                    {TerrainMap[(start_point+35*r + c)]->LookAir(BlueRed,Timer,method);}
                else if (SensType == 3)
                    {TerrainMap[(start_point+35*r + c)]->LookESM(BlueRed,Timer,method);}
            }
        }
    }
}
```

```
// ACCOUNT FOR LOST CONTACT

void TerrainType::UpdateView(const int TheTime) {
    for(int counter = 1; counter <= NumberOfGrids; counter++)    {
        TerrainMap[counter]->UpdateMap(TheTime);
    }
}


//   DEVELOP TARGET LIST

void TerrainType::Target(const int TheTime)   {
    int UN, UTfr, UTtg,AU, AUat, AUms,WepType,Threat,Time,Conf=0,NumAssets;
    double MinDistat, MinDistms,TheDist, Xunit, Yunit, AimptError;
    //method<<endl;
    for (int i = 1; i<= NumberOfGrids;i++) {
            TerrainMap[i]->InitQueue();
            while (TerrainMap[i]->LookAtView(UN,0,Xunit,Yunit,Conf)) {
            // method<<"BLUE[Time = "<<TheTime<<" ] Grid = "<<i<<"  Unit = "<<UN<<endl;
                UTtg = ForceMap[UN]->GetType();
                if (ForceMap[UN]->GetTarget() == 0 && UTtg > 0) {
                    Threat = 0; MinDistat = 807; MinDistms = 807; AUat = 0; AUms = 0;
                    for (int j = 1; j<= NumBlue; j++)    {
                        UTfr = ForceMap[j]->GetType();
                        TheDist = DetDist(j,Xunit,Yunit);

                        //   THREAT DETERMINATION
                        if (UTfr > 0 && UTfr < 10) {
                            if (TheDist<ForceMap[UN]->GetMR()){Threat = 1;}
                        }
                        if (UTtg == 22)    {Threat = 2;}
                        if ( Threat != 1 && UTtg == 24)   {Threat = 2;}

                        //   DETERMINE CLOSEST UNITS THAT HAVE WEAPONS TO SHOOT
                        if (UTfr == 1 && UTtg < 30 && ForceMap[j]->GetNumMS()>Batac
                            && ForceMap[j]->GetMR() >= TheDist) {
                                AUat = (MinDistat > TheDist)?j:AUat;
                                MinDistat = minimum(MinDistat,TheDist);
                        }
                        if (UTfr == 2 && UTtg < 30 && ForceMap[j]->GetNumMS()>Btom
                            && ForceMap[j]->GetMR() >= TheDist) {
                                AUms = (MinDistms > TheDist)?j:AUms;
                                MinDistms = minimum(MinDistms,TheDist);
                        }
                        if (UTfr == 1 && UTtg > 30 && ForceMap[j]->GetNumB() > Bfac
                            && ForceMap[j]->GetBR() >= TheDist) {
                                AUat = (MinDistat > TheDist)?j:AUat;
                                MinDistat = minimum(MinDistat,TheDist);
                        }
                    }

                    //   SELECT WEAPON
                    WepType = BlueWepSel(UN,AUat,AUms,Threat,NumAssets);

                    //   IF A WEAPON HAS BEEN SELECTED, COMPUTE IMPACT TIME, AIMPOINT
                    //   ERROR AND PLACE ON THE TARGET LIST
                    if (WepType > 0)   {
                        if (WepType <= 3) {Time = TheTime +
                            floor(RandNumber->Norm(10,2)+(60*DetDist(UN,1)/SP_Bcm));}
                        if (WepType == 4) {Time = TheTime +
                            floor(RandNumber->Norm(5,1) + (60*DetDist(UN,1)/SP_Bf));}
                        if (WepType >= 5) {Time = TheTime +
                                                floor(60*DetDist(UN,1)/SP_Bat);}


                        AimptError = DetDist(UN,Xunit,Yunit);
```

```cpp
        AimptError = (AimptError <= 10)?exp(-AimptError/10.0):0.2;
        BlueTarget->PUSH(Time,UN,AUat,WepType,NumAssets,AimptError,1.0);
//      method<<"          "<<AUat<<" Shot "<<NumAssets<<" of type "<<WepType
//              <<" will arrive at "<<Time<<endl;
        ForceMap[UN]->SetTarget(1);
      }
    }
  }


  while (TerrainMap[i]->LookAtView(UN,1,Xunit,Yunit,Conf)) {
// method<<"RED[Time = "<<TheTime<<" ] Grid = "<<i<<"   Unit = "<<UN<<endl;
  UTtg = ForceMap[UN]->GetType();
      if (ForceMap[UN]->GetTarget() == 0 && UTtg > 0) {
      Threat = 0; MinDistat = 807; MinDistms = 807; AUat = 0; AUms = 0;
      AU = 0;
      for (int j = (NumBlue + 1); j<= (NumBlue + NumRed); j++) {
        UTfr = ForceMap[j]->GetType();
          TheDist = DetDist(j,Xunit,Yunit);

          // THREAT DETERMINATION
          if (UTfr > 0 && UTfr < 30) {
            if (TheDist <= ForceMap[UN]->GetMR())   {Threat = 1;}
          }

          // DETERMINE CLOSEST UNITS THAT HAVE WEAPONS TO SHOOT
          if (UTfr == 21 && UTtg <10 && ForceMap[j]->GetNumMS()>Ratac
            && ForceMap[j]->GetMR() >= TheDist) {
                AUat = (MinDistat > TheDist)?j:AUat;
                MinDistat = minimum(MinDistat,TheDist);
          }
          if (UTfr == 24 && UTtg < 10 && ForceMap[j]->GetNumMS()> 0
            && ForceMap[j]->GetMR() >= TheDist) {
                AUms = (MinDistms > TheDist)?j:AUms;
                MinDistms = minimum(MinDistms,TheDist);
          }
          if (UTfr == 21 && UTtg > 10 && ForceMap[j]->GetNumB()> Rfac
                && ForceMap[j]->GetBR() >= TheDist) {
                AUat = (MinDistat > TheDist)?j:AUat;
                MinDistat = minimum(MinDistat,TheDist);
          }
      }

      // SELECT WEAPON
      if (AUms > 0)
        {AU = AUms;WepType = RedWepSel(UN,AUms,Threat,NumAssets); }
      else if (AUat > 0)
        {AU = AUat;WepType = RedWepSel(UN,AUat,Threat,NumAssets);}
      else {WepType = 0;}

      // IF A WEAPON HAS BEEN SELECTED, COMPUTE IMPACT TIME, AIMPOINT
      // ERROR AND PLACE ON THE TARGET LIST
      if (WepType > 0) {
        if(WepType <= 13) {Time = TheTime +
            floor(RandNumber->Norm(5,1)+(60*DetDist(UN,1)/SP_Rcm));}
        if (WepType == 14)   {Time = TheTime +
                                    floor(60*DetDist(UN,1)/SP_Rf);}
        if (WepType >= 15)   {Time = TheTime +
            floor(RandNumber->Norm(10,2)+ (60*DetDist(UN,1)/SP_Rat));}
        AimptError = DetDist(UN,Xunit,Yunit);
        AimptError = (AimptError <= 10)?exp(-AimptError/10.0):0.2;
        RedTarget->PUSH(Time,UN,AU,WepType,NumAssets,AimptError,1.0);
//      method<<"          "<<AU<<" Shot "<<NumAssets<<" a "<<WepType
//              <<" will arrive at "<<Time<<endl;
        ForceMap[UN]->SetTarget(1);
      }
  }
```

```cpp
            }
        }
}

//  SELECT A WEAPON TO ATTACK THE RED FORCES

int TerrainType::BlueWepSel( const int UN,int &AUatt, const int AUmsl,
                             const int TH, int &assets)   {
    int TheType = ForceMap[UN]->GetType();


    //  HIGH PRIORITY THREAT


    if (TH == 1)    {
        if (AUatt > 0 && TheType == 21)   {
            assets = BHa < ForceMap[AUatt]->GetNumMS()?BHa:BLa;
            if (assets != BHa)     {
     assets = BLa < ForceMap[AUatt]->GetNumMS()?BLa:ForceMap[AUatt]->GetNumMS();}
            BlueA = BlueA + assets;
            ForceMap[AUatt]->UseMS(assets);
            return 5;
        }
        else if  (AUmsl > 0 && TheType == 21)   {
     assets = BMa < ForceMap[AUmsl]->GetNumMS()?BMa:ForceMap[AUmsl]->GetNumMS();
         BlueCM = BlueCM + assets;
         ForceMap[AUmsl]->UseMS(assets);AUatt = AUmsl;
         return 1;
         }
        else if (AUatt > 0 && TheType == 22)     {
     assets = 6 < ForceMap[AUatt]->GetNumMS()?4:ForceMap[AUatt]->GetNumMS();
         BlueA = BlueA + assets;
         ForceMap[AUatt]->UseMS(assets);
         return 6;
     }
        else if (AUmsl > 0 && TheType == 22)     {
     assets = 6 < ForceMap[AUmsl]->GetNumMS()?6:ForceMap[AUmsl]->GetNumMS();
         BlueCM = BlueCM + assets;
         ForceMap[AUmsl]->UseMS(assets);
         AUatt = AUmsl;
         return 6;
     }
        else if (AUatt > 0 && TheType == 24)     {
      assets = BHs < ForceMap[AUatt]->GetNumMS()?BHs:ForceMap[AUatt]->GetNumMS();
            BlueA = BlueA + assets;
            ForceMap[AUatt]->UseMS(assets);
            return 7;
        }
        else if  (AUmsl > 0 && TheType == 24)   {
       assets = BMs < ForceMap[AUmsl]->GetNumMS()?BMs:ForceMap[AUmsl]->GetNumMS();
         BlueCM = BlueCM + assets;
         ForceMap[AUmsl]->UseMS(assets);
         AUatt = AUmsl;
         return 2;
         }
        else return 0;

    }


    if (TH == 2)    {
        if (AUatt > 0 && TheType == 21)   {
        assets = BLa < ForceMap[AUatt]->GetNumMS()?BLa:0;
            if (assets > 0)    {
                BlueA = BlueA + assets;  ForceMap[AUatt]->UseMS(assets); return 5;
            }
        }
        if (AUmsl > 0 && TheType == 21)   {
            assets = BMa < ForceMap[AUmsl]->GetNumMS()?BMa:0;
```

97

```cpp
        if (assets > 0)      {
            BlueCM = BlueCM + assets;
            ForceMap[AUmsl]->UseMS(assets);
            AUatt = AUmsl;
            return 1;
        }
        else {return 0;}
    }
    if (AUatt > 0 && TheType == 24)   {
        assets = BHs < ForceMap[AUatt]->GetNumMS()?BHs:0;
        if (assets > 0)      {
            BlueA = BlueA + assets;  ForceMap[AUatt]->UseMS(assets); return 7;
        }
    }
    if (AUmsl > 0 && TheType == 24)   {
        assets = BMs < ForceMap[AUmsl]->GetNumMS()?BMs:0;
        if (assets > 0)      {
            BlueCM = BlueCM + assets;
            ForceMap[AUmsl]->UseMS(assets);
            AUatt = AUmsl;
            return 2;
        }
        else {return 0;}
    }


    if (AUatt > 0 && TheType == 22)   {
        assets = BHr < ForceMap[AUatt]->GetNumMS()?BHr:0;
        if (assets > 0)      {
            BlueA = BlueA + assets;   ForceMap[AUatt]->UseMS(assets); return 6;
        }
    }
    if (AUmsl > 0 && TheType == 24)   {
        assets = BMr < ForceMap[AUmsl]->GetNumMS()?BMr:0;
        if (assets > 0)      {
            BlueCM = BlueCM + assets;
            ForceMap[AUmsl]->UseMS(assets);
            AUatt = AUmsl;
            return 2;
        }
        else {return 0;}
    }
    return 0;
}

if (AUatt > 0 && TheType == 33)   {
    assets = BHf < ForceMap[AUatt]->GetNumB()?BHf:0;
    if (assets > 0)      {
        BlueF = BlueF + assets; ForceMap[AUatt]->UseB(assets); return 4;
    }
    else {return 0;}
}
return 0;
}


// SELECT A WEAPON TO ATTACK THE BLUE FORCES

int TerrainType::RedWepSel(const int UN,const int AU,const int TH,int &assets){
    int TheType = ForceMap[UN]->GetType();
    int AvailAssets = ForceMap[AU]->GetNumMS();

    if (TheType == 1 && ForceMap[AU]->GetType() == 21) {
        if (TH == 1)   {assets = RHa < AvailAssets?RHa:AvailAssets;
            RedA = RedA + assets;  ForceMap[AU]->UseMS(assets); return 15;}
        else   {assets = RLa < AvailAssets?RLa:AvailAssets;
            RedA = RedA + assets;ForceMap[AU]->UseMS(assets); return 15;}
    }
```

```
if (TheType == 1 && ForceMap[AU]->GetType() == 24)    {
  assets = AvailAssets;
  ForceMap[AU]->UseMS(assets);
  RedSSM = RedSSM + assets;
  return 11;
}


if (TheType == 2 && ForceMap[AU]->GetType() == 21)    {
  if (TH == 1)    {assets = RHs < AvailAssets?RHs:AvailAssets;
     RedA = RedA + assets;  ForceMap[AU]->UseMS(assets); return 16;}
  else   {assets = RLs < AvailAssets?RLs:AvailAssets;
     RedA = RedA + assets;ForceMap[AU]->UseMS(assets); return 16;}
}


if (TheType == 2 && ForceMap[AU]->GetType() == 24)    {
  assets = RMs < AvailAssets?RMs:AvailAssets;
  ForceMap[AU]->UseMS(assets);
  RedSSM = RedSSM + RMs;
  return 12;
}


if (TheType == 3 && ForceMap[AU]->GetType() == 21)    {
  assets = RLs < AvailAssets?RLs:AvailAssets;
  RedA = RedA + assets;ForceMap[AU]->UseMS(assets); return 17;
}


if (TheType == 3 && ForceMap[AU]->GetType() == 24)    {
  assets = RMs < AvailAssets?RMs:AvailAssets;
  ForceMap[AU]->UseMS(assets);
  RedSSM = RedSSM + assets;
  return 13;
}


if (ForceMap[AU]->GetType() == 21 && (TheType == 12 || TheType == 13)) {
  assets = RHf < ForceMap[AU]->GetNumB()?RHf:ForceMap[AU]->GetNumB();
  ForceMap[AU]->UseB(assets);
  RedF = RedF + assets;
  return 14;
}
return 0;
}


// ADJUDICATE ANY CONFLICTS

void TerrainType::Adjudicate(const int TheClock,ostream &method)    {
  int UN, WepType,Time,Assigned_Unit,NumAssets;
  double AimptErr, dummy, Hits_On_Target;
  int Blue = BlueTarget->stackSize();
  int Red = RedTarget->stackSize();

  for (int i = 1; i<= Blue; i++)    {
    BlueTarget->POP(Time,UN,Assigned_Unit,WepType,NumAssets,AimptErr,dummy);
    if (Time > TheClock) {
      BlueTarget->PUSH(Time,UN,Assigned_Unit,WepType,NumAssets,AimptErr,dummy);
    }
    else {
      Hits_On_Target = BStrike(TheClock,WepType,UN,NumAssets,Assigned_Unit,AimptErr);
      ForceMap[UN]->Attack(Hits_On_Target);
      if (WepType == 1) {ACdamage(UN,Hits_On_Target,1);}
      if (WepType == 5) {ACdamage(UN,Hits_On_Target,2);}
    }
      if (WepType == 4) {ERed_Recon++;}
      if (ForceMap[UN]->Dead() == 1)    {
        UpdateCounter(ForceMap[UN]->GetType());
        ForceMap[UN]->SetType(-1);
```
99

```
            method<<TheClock<<" "<<UN<<endl;
        }
        if (ForceMap[UN]->Dead() == 0) {ForceMap[UN]->SetTarget(0);}
    }

    for (i = 1; i<= Red; i++)  {
        RedTarget->POP(Time,UN,Assigned_Unit,WepType,NumAssets,AimptErr,dummy);
        if (Time > TheClock) {
            RedTarget->PUSH(Time,UN,Assigned_Unit,WepType,NumAssets,AimptErr,dummy);
        }
        else {
            Hits_On_Target = RStrike(TheClock,WepType,UN,NumAssets,Assigned_Unit,AimptErr);
            ForceMap[UN]->Attack(Hits_On_Target);
            if (WepType == 11)    {ACdamage(UN,Hits_On_Target,1);}
            if (WepType == 15)    {ACdamage(UN,Hits_On_Target,2);}
        }
        if (WepType == 14)    {EBlue_Recon++;}
        if (ForceMap[UN]->Dead() == 1)    {
            UpdateCounter(ForceMap[UN]->GetType());
            ForceMap[UN]->SetType(-1);
            method<<TheClock<<" "<<UN<<endl<<endl;
        }
        if (ForceMap[UN]->Dead() == 0){ForceMap[UN]->SetTarget(0);}
    }
}


//   DETERMINE ASSETS LOST AND HITS RECORDED FOR A BLUE ATTACK AGAINST RED

int TerrainType::BStrike(const int PresTime,const int WT, const int UN,const int NA, cons
t int AU,const double Err)   {
    int RetTime,Strikes = 0;
    int Lost = 0;

    if (WT == 1)   {
        if (ForceMap[UN]->GetType() > 0) {
            Strikes = Fight(Ph_Bcm_Rab*Err,Pd_Bcm_Rab,NA,Lost);
            BlueCMLoss = BlueCMLoss + Lost;
        }
        return Strikes;
    }
    if (WT == 2)    {
        if (ForceMap[UN]->GetType() > 0) {
            Strikes =Fight(Ph_Bcm_Rradar*Err,Pd_Bcm_Rradar,NA,Lost);
            BlueCMLoss = BlueCMLoss + Lost;
        }
        return Strikes;
    }
    if (WT == 3)    {
        if (ForceMap[UN]->GetType() > 0) {
            Strikes =Fight(Ph_Bcm_RSSM*Err,Pd_Bcm_RSSM,NA,Lost);
            BlueCMLoss = BlueCMLoss + Lost;
        }
        return Strikes;
    }
    if (WT == 4)    {
        if (ForceMap[UN]->GetType() > 0) {
            Strikes = Fight(Ph_Bf_Rrecon*Err,Pd_Bf_Rrecon,NA,Lost);
            BlueFLoss = BlueFLoss + Lost;
        }
        RetTime = PresTime + floor(RandNumber->Norm(90,30) + (60*DetDist(UN,AU)/SP_Bf));
        RetAircraft->PUSH(RetTime,AU,UN,2,NA-Lost,1.0,1.0);
        return (Strikes > 0)?1:0;
    }
    if (WT == 5)    {
        if (ForceMap[UN]->GetType() > 0) {
```

100

```
            Strikes = Fight(Ph_Bat_Rab*Err,Pd_Bat_Rab,NA,Lost);
            BlueALoss = BlueALoss + Lost;
        }
        RetTime = PresTime + floor(RandNumber->Norm(90,30) + (60*DetDist(UN,AU)/SP_Bf));
        RetAircraft->PUSH(RetTime,AU,UN,1,NA-Lost,1.0,1.0);
        return Strikes;
    }
    if (WT == 6)   {
        if (ForceMap[UN]->GetType() > 0) {
            Strikes = Fight(Ph_Bat_Rradar*Err,Pd_Bat_Rradar,NA,Lost);
            BlueALoss = BlueALoss + Lost;
        }
        RetTime = PresTime + floor(RandNumber->Norm(90,30) + (60*DetDist(UN,AU)/SP_Bf));
        RetAircraft->PUSH(RetTime,AU,UN,1,NA-Lost,1.0,1.0);
        return Strikes;
    }
    if (WT == 7)   {
        if (ForceMap[UN]->GetType() > 0) {
            Strikes = Fight(Ph_Bat_RSSM*Err,Pd_Bat_RSSM,NA,Lost);
            BlueALoss = BlueALoss + Lost;
        }
        RetTime = PresTime + floor(RandNumber->Norm(90,30) + (60*DetDist(UN,AU)/SP_Bf));
        RetAircraft->PUSH(RetTime,AU,UN,1,NA-Lost,1.0,1.0);
        return Strikes;
    }
    return 0;
}


//  DETERMINE ASSETS LOST AND HITS RECORDED FOR A RED ATTACK AGAINST BLUE

int TerrainType::RStrike(const int PresTime,const int WT, const int UN, const int NA, con
st int AU,const double Err) {
    int Strikes= 0;
    int RetTime,Lost = 0;
    if (WT == 11)   {
        if (ForceMap[UN]->GetType() > 0) {
            Strikes = Fight(Ph_RSSM_Bcar*Err,Pd_RSSM_Bcar,NA,Lost);
            RedSSMLoss = RedSSMLoss + Lost;
        }
        return Strikes;
    }
    if (WT == 12)   {
        if (ForceMap[UN]->GetType() > 0) {
            Strikes =Fight(Ph_RSSM_Bwar*Err,Pd_RSSM_Bwar,NA,Lost);
            RedSSMLoss = RedSSMLoss + Lost;
        }
        return Strikes;
    }
    if (WT == 13)   {
        if (ForceMap[UN]->GetType() > 0) {
            Strikes =Fight(Ph_RSSM_Bmerch*Err,Pd_RSSM_Bmerch,NA,Lost);
            RedSSMLoss = RedSSMLoss + Lost;
        }
        return Strikes;
    }
    if (WT == 14)   {
        if (ForceMap[UN]->GetType() > 0) {
            Strikes = Fight(Ph_Rf_Brecon*Err,Pd_Rf_Brecon,NA,Lost);
            RedFLoss = RedFLoss + Lost;
        }
        RetTime = PresTime + floor(RandNumber->Norm(90,30) + (60*DetDist(UN,AU)/SP_Bf));
        RetAircraft->PUSH(RetTime,AU,UN,2,NA-Lost,1.0,1.0);
        return (Strikes > 0)?1:0;
    }
    if (WT == 15)   {
        if (ForceMap[UN]->GetType() > 0) {
```

```
                Strikes = Fight(Ph_Rat_Bcar*Err,Pd_Rat_Bcar,NA,Lost);
                RedALoss = RedALoss + Lost;
            }
            RetTime = PresTime + floor(RandNumber->Norm(90,30) + (60*DetDist(UN,AU)/SP_Bf));
            RetAircraft->PUSH(RetTime,AU,UN,1,NA-Lost,1.0,1.0);
            return Strikes;
        }
        if (WT == 16)  {
            if (ForceMap[UN]->GetType() > 0) {
                Strikes = Fight(Ph_Rat_Bwar*Err,Pd_Rat_Bwar,NA,Lost);
                RedALoss = RedALoss + Lost;
            }
            RetTime = PresTime + floor(RandNumber->Norm(90,30) + (60*DetDist(UN,AU)/SP_Bf));
            RetAircraft->PUSH(RetTime,AU,UN,1,NA-Lost,1.0,1.0);
            return Strikes;
        }
        if (WT == 17)  {
            if (ForceMap[UN]->GetType() > 0) {
                Strikes = Fight(Ph_Rat_Bmerch*Err,Pd_Rat_Bmerch,NA,Lost);
                RedALoss = RedALoss + Lost;
            }
            RetTime = PresTime + floor(RandNumber->Norm(90,30) + (60*DetDist(UN,AU)/SP_Bf));
            RetAircraft->PUSH(RetTime,AU,UN,1,NA-Lost,1.0,1.0);
            return Strikes;
        }
        return 0;
}


//   RETURN AIRCRAFT TO SERVICE THAT HAVE LANDED AND RELOADED

void TerrainType::ReturnAircraft(const int TheClock)  {
    int UN, WepType,Time,dummy1,NumAssets;
    double dummy2,dummy3;
    int Size = RetAircraft->stackSize();

    for (int i = 1; i<= Size; i++)    {
        RetAircraft->POP(Time,UN,dummy1,WepType,NumAssets,dummy2,dummy3);
        if (Time > TheClock) {
            RetAircraft->PUSH(Time,UN,dummy1,WepType,NumAssets,dummy2,dummy3);
        }
        else {
            if (WepType == 1) {ForceMap[UN]->UseMS(-NumAssets);}
            if (WepType == 2) {ForceMap[UN]->UseB(-NumAssets);}
        }
    }
}




//  PRINT OUT THE STATUS OF ALL UNITS AT A GIVEN TIME

 void TerrainType::Status(ostream &method, int TheTime)   {
    method<<TheTime<<endl;
    UnitPrint(method);
    }

//   OUPUT WEAPONS USED DURING SIMULATION

void TerrainType::WepsStatus(ostream &method)    {
    method<<endl<<"Blue"<<endl
            <<"CM:      "<<BlueCM<<" "<<BlueCMLoss<<endl
            <<"A A/C:   "<<BlueA<<" "<<BlueALoss<<endl
            <<"F A/C:   "<<BlueF<<" "<<BlueFLoss<<endl
            <<"Red"<<endl
```

102

```
              <<"SSM:      "<<RedSSM<<" "<<RedSSMLoss<<endl
              <<"A A/C:    "<<RedA<<" "<<RedALoss<<endl
              <<"F A/C:    "<<RedF<<" "<<RedFLoss<<endl;
}


//   DETERMINE DISTANCE BETWEEN TWO UNITS

  double  TerrainType::DetDist(const  int  FirstUnit,  const  int  SecondUnit)  {
     double  xcomp,  ycomp;
     xcomp = ForceMap[FirstUnit]->GetxCoord() - ForceMap[SecondUnit]->GetxCoord();
     ycomp = ForceMap[FirstUnit]->GetyCoord() - ForceMap[SecondUnit]->GetyCoord();
     return  sqrt(xcomp*xcomp + ycomp*ycomp);
  }

  double  TerrainType::DetDist(const  int  FirstUnit,const  double  X,const  double  Y){
     double  xcomp,  ycomp;
     xcomp = ForceMap[FirstUnit]->GetxCoord() - X;
     ycomp = ForceMap[FirstUnit]->GetyCoord() - Y;
     return  sqrt(xcomp*xcomp + ycomp*ycomp);
  }


//   DETERMINE THE GRID THAT A UNIT IS LOCATED ON

int  TerrainType::DetermineGrid(const  int  Unit)   {
     double  x,y;
     int  counter = 1;
     x = ForceMap[Unit]->GetxCoord();
     y = ForceMap[Unit]->GetyCoord();
     while  (y>TerrainMap[counter]->GetyUp() && y >TerrainMap[counter]->GetyDown())
         {counter = counter+5;}
     while  (x>TerrainMap[counter]->GetxLeft()&&x>TerrainMap[counter]->GetxRight())
         {counter++;}
     return  counter;
}



//   ADJUDICATE A CONFLICT BETWEEN TWO UNITS

int  TerrainType::Fight(const  double  Ph,const  double  Pd,const  int  NC,int  &Losses)
        {
        int  Strike = 0;
        Losses = 0;
        for  (int i = 1; i <= NC; i++) {
           if  (RandNumber->Unif(0,1) > Pd)   {
              if(RandNumber->Unif(0,1) < Ph)    {Strike++;}
           }
           else  {Losses++;}
        }
        return  Strike;
}


//   KEEP TRACK OF TOTAL FORCE UNIT LOSSES AFTER A CONFLICT ADJUDICATION
void  TerrainType::UpdateCounter(const  int  UNtype)   {
    if      (UNtype == 1)    {ECarrier++;}
    else if (UNtype == 2)    {EWarship++;}
    else if (UNtype == 3)    {EMerchant++;}
    else if (UNtype == 21)   {EAirbase++;}
    else if (UNtype == 22)   {ELong_Range_Radar++;}
    else if (UNtype == 24)   {ESSM++;}
}


//   RECORD THE TYPE AND NUMBER OF EACH TYPE OF UNIT PRIOR TO BATTLE COMMENCEMENT

void  TerrainType::InitCounter(const  int  UNtype,const  int  HTK,const  int  Att,
                                                      const  int  F)     {
    if        (UNtype == 1)    {
```
103

```
            ICarrier++;
            IBlueA = IBlueA + Att;
            IBlueF = IBlueF + F;
    }
    else if (UNtype == 2)    {IWarship++;}
    else if (UNtype == 3)    {IMerchant++;}
    else if (UNtype == 12 || UNtype == 13) {IBlue_Recon = IBlue_Recon + HTK; }
    else if (UNtype == 21)   {
            IAirbase++;
            IRedA = IRedA + Att;
            IRedF = IRedF + F;
    }
    else if (UNtype == 22)   {ILong_Range_Radar++;}
    else if (UNtype == 24)   {ISSM++;}
    else if (UNtype == 33)   {IRed_Recon = IRed_Recon + HTK;}
}


void TerrainType::ACdamage(const int Tget, const int Hits, const int MorB) {
    int TotAc,ACdown,Att;
    double Percent, Effectiveness;

    TotAc = ForceMap[Tget]->GetNumMS() + ForceMap[Tget]->GetNumB();
    Effectiveness = RandNumber->Unif(0,2);

        if (Tget < 20) { Percent = (MorB == 1)?USMsl:USB;}
        else   {            Percent = (MorB == 1)?IRMsl:IRB;}

        ACdown = floor(double (Hits)*Percent*Effectiveness*double (TotAc));
        Att = floor(double (ACdown)*double (ForceMap[Tget]->GetNumMS())/double (TotAc));
        Att = (ForceMap[Tget]->GetNumMS() > Att)?Att:ForceMap[Tget]->GetNumMS();
        ForceMap[Tget]->UseMS(Att);
        Att = ACdown - Att;
        Att = (ForceMap[Tget]->GetNumB() > Att)?Att:ForceMap[Tget]->GetNumB();
        ForceMap[Tget]->UseB(Att);
    }
```

```
//   Edward R. Martinez
//   September 1996
//   MSDOS 6.22
//   Borland C++ 4.02 for windows
//   THIS IS FILE Grid.h

//   This class models a 20 square mile grid of terrain
//   The file Grid.cpp holds the code for the class.

#ifndef __Grid_h
#define __Grid_h
#include <iostream.h>
#include <fstream.h>
#include "Grid.h"
#include "queue.h"
#include "stack.h"
#include "RdNumGen.h"

//DEFINE CLASS GridType

class GridType {
    public :
        GridType();
        ~GridType();
        void UpdateMap(const int);
        void SetGridPoints(int, int, int, int, int, int);
        void AddUnit(const int, const int, const int, const double, const double);
        void RemoveUnit(int);
        void GridPrint(ostream&);
        void LookSurf(int,const int,ostream&);
        void LookAir(int,const int,ostream&);
        void LookESM(int,const int,ostream&);
        void Sensor_Process(const int, const int, const int,const int, const double);
        int LookAtView(int&, const int, double&, double&, int&);
        void SetWeather(int NewValue) {Weather = NewValue;}
        double GetWeather(){return Weather;}
        float GetxLeft()    {return xLeft;}
        float GetxRight()  {return xRight;}
        void InitQueue()   {BlueView->InitLookPtr(); RedView->InitLookPtr();}
        float GetyUp() {return yUp;}
        int GetType()    {return Type_of_Grid;}
        float GetyDown()   {return yDown;}
        int AnyUnits();

    private :
        int xLeft, xRight, yUp, yDown;
        int Weather, Type_of_Grid;
        RdGen *RandNum;
        queueType *Blue, *Red, *BlueView, *RedView;

        //   STATIC VARIABLES
        static double ASurfB, AAirB, AESMB, ASurfR, AAirR, AESMR;
};
#endif
```

```
//   Edward R. Martinez
//   THESIS
//   September 1996
//   MSDOS 6.2
//   Borland C++ 4.02 for windows
//   THIS IS FILE Grid.cpp
//
//   This class models a Grid on a Terrain Map
//   This class is utilized by class TerrainType in the file Terrain.cpp

#include <iostream.h>
#include <fstream.h>
#include "RdNumGen.h"
#include "Grid.h"
#include "queue.h"

//   DEFINE STATIC VARIABLES

    double  GridType::ASurfB = 0;
    double  GridType::AAirB = 0;
    double  GridType::AESMB = 0;
    double  GridType::ASurfR = 4;
    double  GridType::AAirR = .5;
    double  GridType::AESMR = 4;

//   CONSTRUCTOR

    GridType::GridType() {
        Blue = new queueType();
        Red = new queueType();
        BlueView = new queueType();
        RedView = new queueType();
        RandNum = new RdGen();
    }

//   DESTRUCTOR

    GridType::~GridType()   {
        delete Blue;
        delete Red;
        delete BlueView;
        delete RedView;
        delete RandNum;
    }

//   UPDATE INTELLIGENCE PERCEPTION

void GridType::UpdateMap(const int PresentTime) {
    int UN, Qsize;
    BlueView->InitLookPtr();
    if (!BlueView->queueEmpty())   {
        Qsize = BlueView->queueSize();
        for (int i = 0; i < Qsize; i++)   {
            if ((PresentTime - BlueView->GetLookTime()) > 15)   {
                UN = BlueView->LookUnit();
                BlueView->IncLookPtr();
                BlueView->RemoveMember(UN);
            }
            else   {BlueView->IncLookPtr();}
        }
    }
    RedView->InitLookPtr();
    if (!RedView->queueEmpty())     {
        Qsize = RedView->queueSize();
        for (int j = 0; j < Qsize; j++)   {
            if ((PresentTime - RedView->GetLookTime()) > 15)     {
```
106

```
                UN = RedView->LookUnit();
                RedView->IncLookPtr();
                RedView->RemoveMember(UN);
            }
            else  {RedView->IncLookPtr();}
        }
    }
}


//  INITIALIZE A GRID

void GridType::SetGridPoints(int xL, int xR, int yU, int yD, int t, int w) {
    xLeft    = xL;
    xRight   = xR;
    yUp      = yU;
    yDown    = yD;
    Type_of_Grid = t;
    Weather = w;
}


//  ADD A UNIT TO A GRID

    void GridType::AddUnit(const int Unit_Number, const int Number,
                    const int TheTime, const double XNew, const double YNew)     {
        if (Unit_Number < 21) {
            Blue->AddFront(Unit_Number,Number, TheTime, XNew, YNew);
        }
        if (Unit_Number > 20) {
            Red->AddFront(Unit_Number,Number, TheTime,XNew,YNew);
        }
    }


//  REMOVE A UNIT FROM A GRID

    void GridType::RemoveUnit(int Unit_Number)   {
        if (Unit_Number < 21) {
            Blue->RemoveMember(Unit_Number);
        // BlueView->RemoveMember(Unit_Number);
        }
        if (Unit_Number > 20) {
            Red->RemoveMember(Unit_Number);
        // RedView->RemoveMember(Unit_Number);
        }
    }

//  PRINT OUT A LISTING OF GRIDS

    void GridType::GridPrint(ostream &method) {
            if (Blue->queueSize() > 0 || Red->queueSize() > 0) {
            method<<xLeft<<" "<<xRight<<" "<<yUp<<" "<<yDown
                <<" "<<Type_of_Grid<<" Num Blue = "<<Blue->queueSize()
                <<" Red Saw = "<<RedView->queueSize()
                <<" Num Red = "<<Red->queueSize()
                <<" Blue Saw = "<<BlueView->queueSize();}
    }

    int GridType::AnyUnits()    {
        if (Blue->queueSize() > 0 || Red->queueSize() >0)   {return 1;}
        return 0;
    }

//  SEARCH USING SURFACE SEARCH RADAR

void GridType::LookSurf(int Color, const int DeTime,ostream &meth)    {
    int UnitNumb, Emit;
    if (Color == 0)   {
```

107

```cpp
        Red->InitLookPtr();
        if (!Red->queueEmpty()) {
            for (int i = 0; i < Red->queueSize(); i++)    {
                UnitNumb = Red->LookAtGrid(Emit);
                if (UnitNumb < 30 && RandNum->Unif(0,1) >.3 )    {
                    Sensor_Process(0, DeTime, UnitNumb, Emit,ASurfB);
                    meth<<" ["<<UnitNumb<<"] ";
                }
                Red->IncLookPtr();
            }
        }
    }
    else    {
        Blue->InitLookPtr();
        if (!Blue->queueEmpty())    {
            for (int i = 0; i < Blue->queueSize(); i++)  {
                UnitNumb = Blue->LookAtGrid(Emit);
                if (UnitNumb < 10 && RandNum->Unif(0,1) > .3)    {
                    Sensor_Process(1, DeTime, UnitNumb, Emit,ASurfR);
                    meth<<" ["<<UnitNumb<<"] ";
                }
                Blue->IncLookPtr();
            }
        }
    }
}


//   SEARCH USING AIR SEARCH RADAR

void GridType::LookAir(int Color, const int DeTime, ostream & meth)  {
    int UnitNumb, Emit;

    if (Color == 0)    {
        Red->InitLookPtr();
        if (!Red->queueEmpty()) {
            for (int i = 0; i < Red->queueSize(); i++)    {
                UnitNumb = Red->LookAtGrid(Emit);
                if (UnitNumb > 30 && RandNum->Unif(0,1) >.2) {
                    Sensor_Process(0, DeTime, UnitNumb, Emit,AAirB);
                    meth<<" ["<<UnitNumb<<"] ";
                }
                Red->IncLookPtr();
            }
        }
    }
    else    {
    Blue->InitLookPtr();
        if (!Blue->queueEmpty())    {
            for (int i = 0; i < Blue->queueSize(); i++)  {
                UnitNumb = Blue->LookAtGrid(Emit);
                if (UnitNumb > 10 && UnitNumb != 16 && UnitNumb != 17 &&
                    UnitNumb !=18 && UnitNumb != 19 && RandNum->Unif(0,1) >.2){
                    Sensor_Process(1, DeTime, UnitNumb, Emit,AAirR);
                    meth<<" ["<<UnitNumb<<"] ";
                }
                Blue->IncLookPtr();
            }
        }
    }
}

// SEARCH USING ELINT RECEIVERS

void GridType::LookESM(int Color, const int DeTime,ostream &meth) {
    int UnitNumb, Emit;
    if (Color == 0)    {
```

```
                Red->InitLookPtr();
                if (!Red->queueEmpty()) {
                    for (int i = 0; i < Red->queueSize(); i++)    {
                        UnitNumb = Red->LookAtGrid(Emit);
                        if ( Emit == 1 && RandNum->Unif(0,1) > .4)    {
                            Sensor_Process(0, DeTime, UnitNumb, Emit,AESMB);
                            meth<<" ["<<UnitNumb<<"] ";
                        }
                        Red->IncLookPtr();
                    }
                }
            }
        else    {
            Blue->InitLookPtr();
            if (!Blue->queueEmpty())    {
                for (int i = 0; i < Blue->queueSize(); i++)   {
                    UnitNumb = Blue->LookAtGrid(Emit);
                    if (Emit == 1 && UnitNumb != 16 && UnitNumb != 17 &&
                        UnitNumb != 18 && UnitNumb != 19 && RandNum->Unif(0,1) > .4){
                        Sensor_Process(1, DeTime, UnitNumb, Emit,AESMR);
                        meth<<" ["<<UnitNumb<<"] ";
                    }
                    Blue->IncLookPtr();
                }
            }
        }
    }
}


//  SENSOR PROCESS

 void GridType::Sensor_Process(const int CL, const int DT, const int UN,
                               const int NU, const double ACC)   {
    double X,Y;

    if (CL == 0)    {Red->LookUpXY(X,Y);}
    else {Blue->LookUpXY(X,Y);}

    X = RandNum->Norm(X,ACC);
    if (X < 0)    {X = 0;}
    if (X > 700) {X = 700;}
    Y = RandNum->Norm(Y,ACC);
    if (Y < 0)   {Y = 0;}
    if (Y > 400) {Y = 400;}

    if (CL == 0)    {
        if (BlueView->FindElement(UN))    {
            BlueView->SecondSight(DT,X,Y);
        }
        else {BlueView->AddFront(UN,NU,DT,X,Y);}
    }
    else    {
        if (RedView->FindElement(UN)) {
            RedView->SecondSight(DT,X,Y);
        }
        else {RedView->AddFront(UN,NU,DT,X,Y);}
    }
}



//  LOOK AT VIEW

int GridType::LookAtView(int &Un, const int Cl, double &X, double &Y,int &C){
    if (Cl == 0)    {
        if (BlueView->queueEmpty())    {return 0;}
        if (BlueView->NoLookAvail())   {return 0;}
        Un = BlueView->LookUnit();
```

```
            C = BlueView->LookConf();
            BlueView->LookUpXY(X,Y);
            BlueView->IncLookPtr();
            return 1;
    }
    if (Cl == 1)     {
            if (RedView->queueEmpty()) {return 0;}
            if (RedView->NoLookAvail())    {return 0;}
            Un = RedView->LookUnit();
            C = RedView->LookConf();
            RedView->LookUpXY(X,Y);
            RedView->IncLookPtr();
            return 1;
    }
    return 0;
}
```

```cpp
//   Edward R. Martinez
//   September 1996
//   THESIS
//   MSDOS 6.22
//   Borland C++ 4.02 for windows
//   THIS IS FILE Unit.h


//   This class models a naval or ground combat unit
//   The file Unit.cpp holds the code for the class.
//   This class is used by terrain.cpp


#ifndef __Unit_h
#define __Unit_h

#include <iostream.h>
#include <fstream.h>
#include <math.h>
#include <stdlib.h>
#include "RdNumGen.h"


//DEFINE CLASS UnitType

class UnitType {
    public :
        UnitType();
        void  SetUnit(const int,const int,const int,const int,const int,const int,
        const int,const int,const int,const int,const int,const int,const int,
        const int, const double,const double,const double,const double,
        const double,const double,const double,const double,const double,
        const double,const double,const double,const double,const double,
        const double,const double,const double,const double);
        int MoveSSM(const int,const int, const double);
        int MoveShip(const int);
        int MoveAir(const int,const double, const double, const int,const int,
                                                      const int);
        void UpdateMotion(const double);
        void PrintUnit(ostream&);
        void PrintLoc(ostream&);
        void ChangeSSM();
        void NewSSMLoc();

        double GetxCoord()    {return  xLocation;}
        double GetyCoord()    {return  yLocation;}
        int GetNumber()    {return  Number;}
        double GetCourse(){return  Course;}
        int GetLP5()    {return  LP[5];}
        int GetLP6()    {return  LP[6];}
        void SetType(const int val)    {Type = val;}
        int GetType()    {return  Type;}
        int GetGrid()    {return  GridLocation;}
        int GetSRR()    {return  Surf_Range;}
        int GetARR()    {return  Air_Range;}
        int GetESMR()    {return  ESM_Range;}
        void SetSRR(const int range)    {Surf_Range = range;}
        void SetARR(const int range)    {Air_Range = range;}
        double GetMR() {return  Missile_Range;}
        double GetBR() {return  Bomb_Range;}
        int GetNumMS() {return  Missiles;}
        int GetNumB()    {return  Bombs;}
        void UseMS(const int val)    {Missiles = Missiles - val;}
        void UseB(const int val)    {Bombs = Bombs - val;}
        void ReloadMS(const int val)    {Missiles = Missiles + val;}
        void ReloadB(const int val)    {Bombs = Bombs + val;}
        void SpecifyGrid(int GridLoc) {GridLocation = GridLoc;}
```
111

```cpp
    int GetEMCON() {return Emitting;}
    void SetTarget(const int val) {Target = val;}
    int GetTarget()    {return Target;}
    int Dead()  {return Hits < Hits_To_Kill?0:1;}
    void Attack(const int Num) {Hits = Hits + Num;}

private :

    // VARIABLES

    int  UnitNumber,Number,Type,GridLocation, WP, Motion;
    double  Course, Speed, MaxSpeed,MaxRange, xLocation, yLocation;
    double  XP[7], YP[7];
    int  LP[7];
    int  Missiles, Bombs, Surf_Range, Air_Range, ESM_Range;
    double  Missile_Range, Bomb_Range;
    int  Emitting, Target, Hits, Hits_To_Kill,StopTime;
    RdGen RandNumb;

    // FUNCTIONS

    double  CheckBoundsX(int&, const double );
    double  CheckBoundsY(int&, const double );
    double  CalcCourse(double , double );
    double  Distance();
    void ChartCourse(const int );
};
#endif
```

```
//  Edward R. Martinez
//  THESIS
//  September 1996
//  MSDOS 6.2
//  Borland C++ 4.02 for windows
//  THIS IS FILE Unit.cpp
//
//  This class models a unit in a combat scenario

#include <iostream.h>
#include <fstream.h>
#include <math.h>
#include <stdlib.h>
#include <string.h>
#include "Unit.h"
#include "RdNumGen.h"


//      CONSTRUCTOR

UnitType::UnitType() {
    UnitNumber = 0;
    Number = 1;
    Type = -1;
    GridLocation = 0;
    WP = 1;
    Motion = 0;
    Course = 0.0;
    Speed = 0.0;
    MaxSpeed = 0.0;
    xLocation = 0.0;
    yLocation = 0.0;
    XP[0] = 0.0;XP[1] = 0.0;XP[2] = 0.0;XP[3] = 0.0;XP[4] = 0.0;XP[5] = 0.0;
    XP[6] = 0.0;
    YP[0] = 0.0;YP[1] = 0.0;YP[2] = 0.0;YP[3] = 0.0;YP[4] = 0.0;YP[5] = 0.0;
    YP[6] = 0.0;
    LP[0] = 0;LP[1] = 0;LP[2] = 0;LP[3] = 0;LP[4] = 0;LP[5] = 0;LP[6] = 0;
    Missiles = 0;
    Bombs = 0;
    Surf_Range = 0;
    Air_Range = 0;
    ESM_Range = 0;
    Missile_Range = 0.0;
    Bomb_Range = 0.0;
    Emitting = 1; Target = 0; Hits = 0; Hits_To_Kill = 0;
    StopTime = 1;
}

void UnitType::SetUnit(const int UU,const int UH,const int UT,const int USRR,
    const int UARR,const int UESMR,const int UM,const int UB,const int ULP1,
    const int ULP2,const int ULP3,const int ULP4,const int ULP5, const int ULP6,
    const double UMS,const double UMR,const double UMSR,const double UBR,
    const double UXP0,   const double UYP0,const double UXP1,const double UYP1,
    const double UXP2,const double UYP2,const double UXP3,const double UYP3,
    const double UXP4,const double UYP4, const double UXP5,const double UYP5,
    const double UXP6, const double UYP6)  {

            UnitNumber = UU;
            Hits_To_Kill = UH;
            Type = UT;
            xLocation = UXP0;
            yLocation = UYP0;
            MaxSpeed = UMS;
            MaxRange = UMR;
            Missiles = UM;
            Missile_Range = UMSR;
            Bombs = UB;
```
113

```
                    Bomb_Range = UBR;
                    Surf_Range = USRR;
                    Air_Range = UARR;
                    ESM_Range = UESMR;
                    XP[1] = UXP1; YP[1] = UYP1; LP[1] = ULP1;
                    XP[2] = UXP2; YP[2] = UYP2; LP[2] = ULP2;
                    XP[3] = UXP3; YP[3] = UYP3; LP[3] = ULP3;
                    XP[4] = UXP4; YP[4] = UYP4; LP[4] = ULP4;
                    XP[5] = UXP5; YP[5] = UYP5; LP[5] = ULP5;
                    XP[6] = UXP6; YP[6] = UYP6; LP[6] = ULP6;
}


int UnitType::MoveSSM(const int TheTime,const int Flag, const double pmove)    {

//  Determine if a SSM has reached the next waypoint, if it has determine
//  if it will loiter at this point

    if  (WP == 2)   {
        if (Motion == 1)  {
            if (TheTime == StopTime)   {
                StopTime = TheTime + LP[2];
                Missiles = 16;
                return  0;
            }
            else return 1;
        }
        else  {
            if (TheTime == StopTime)    {
                NewSSMLoc();
                ChartCourse(TheTime);
                WP = 1;
                return 1;
            }
            return  0;
        }
    }

    if (Missiles == 0)    {
            WP = 2;
            ChartCourse(TheTime);
            return 1;
    }

    if (WP == 1)     {
        if (Motion == 1)   {
          if (TheTime == StopTime) {
                Motion = 0;
                Course = 0;
                Speed = 0;
                return  0;
          }
          else {return 1;}
        }
    }

    if (Flag == 1 && RandNumb.Unif(0,1) < pmove) {
        NewSSMLoc();
        ChartCourse(TheTime);
        Emitting = 0;
        return 1;
    }
    if (Flag == 1){Emitting = (RandNumb.Unif(0,1) < 0.5)?0:1;}
    return  0;
}

//  ALL NAVAL VESSEL MOVEMENT
```

```
int UnitType::MoveShip(const int TheTime) {

//  SHIP IS NOT MOVING  1) Done Loitering, pull anchor 2) Continue Loitering

    if (Motion == 0)  {
       if (TheTime == StopTime)    {
          ChartCourse(TheTime);
          return 1;
       }
       else   {return 0;}
    }


//  SHIP IS MOVING 1) Reached destination, loiter or new course
//                 2) Maintain  present course and speed

    if (Motion == 1)  {
       if (TheTime == StopTime)    {
          WP++;
          if (LP[WP-1] > 0) {
             StopTime = TheTime + LP[WP-1];
             Motion = 0;
             return 0;
          }
          else {
             ChartCourse(TheTime);
             return 1;
          }
       }
       else   {return 1;}
    }
    return 1;
}


//  RECONAISSANCE AIRCRAFT MOVEMENT

int UnitType::MoveAir(const int TheTime,const double X, const double Y,
                      const int dirX, const int dirY, const int Color)  {

    if (WP == 1)    {
       if (Motion == 0)  {ChartCourse(TheTime);Emitting = 1;}
       else   {
          if (TheTime == StopTime)    {
             WP = 2;
             ChartCourse(TheTime);
          }
       }
       Motion = 1;
       return 1;
    }

    if (WP == 2)    {
       if (TheTime == StopTime)    {
          WP = 3;
          XP[3] = X;
          YP[3] = Y;
          ChartCourse(TheTime);
       }
       Motion = 1;
       return 1;
    }

    if (WP == 3)    {
       if (TheTime == StopTime)    {
          WP = 4;
          xLocation = X;
```

```
            yLocation = Y;
            Motion = 0;
            Emitting = 0;
            StopTime = TheTime + LP[4];
            return 0;
        }
        XP[3] = X;
        YP[3] = Y;
        ChartCourse(TheTime);
        Motion = 1;
        return 1;
    }

    if (WP == 4)    {
        if (TheTime == StopTime)    {
            WP = 1;
            YP[1] = Y +(dirY*YP[5]);
            XP[1] = X + (dirX*XP[5]);
            YP[2] = YP[1];
            if (Color == 0)    {XP[2] = XP[1] + (dirX*XP[6]);}
            else {XP[2] = XP[1] - (dirX*XP[6]);}
            Motion = 0;
        }
        xLocation = X;
        yLocation = Y;
        return 2;
    }
    return 2;
}


//   UPDATE LOCATION OF A UNIT THAT HAS MOVED

void UnitType::UpdateMotion(const double deltaT)    {
        int dummy = 0;
        xLocation = CheckBoundsX(dummy,cos(Course)*Speed*deltaT + xLocation);
        yLocation = CheckBoundsY(dummy,sin(Course)*Speed*deltaT + yLocation);
    }


// PRINT OUT INFORMATION DESCRIBING A UNIT

void UnitType::PrintUnit(ostream &method) {
    method<<UnitNumber<<" "<<Type<<" HITS/HTK = ("<<Hits<<","
        <<Hits_To_Kill<<") "
        <<"# Msl =   "<<Missiles<<" "
        "Location = ("<<xLocation<<","<<yLocation<<")"
        <<"Grid = ["<<GridLocation<<"]"
        <<endl;
}


//   PRINT OUT COORDINATES TO DATA FILE FOR DISPLAY PURPOSES

void UnitType::PrintLoc(ostream &method)    {
    method<<UnitNumber<<" "<<xLocation<<" "<<yLocation<<endl;
}


//   ENSURE SSM STAYS ON LAND

void UnitType::ChangeSSM() {
    int flag = 0;
    XP[1] = CheckBoundsX(flag, 10 + xLocation);
    if (flag == 1) {
        if (XP[1] == 0)    {XP[1] = XP[1] + 5;}
        else {XP[1] = XP[1] - 5;}
    }
    flag = 0;
    YP[1] = CheckBoundsY(flag, 10 - yLocation);
```

```
      if (flag == 1) {
         if (YP[1] == 0)    {YP[1] = YP[1] + 5;}
         else {YP[1] = YP[1] - 5;}
      }
}


//   ENSURE A UNIT DOES NOT EXCEED X LIMITS OF GRID MAP

double UnitType::CheckBoundsX(int &flag, const double Location)    {
   double temp = Location;
   if (Location <0) {    temp = 0;    flag = 1;}
   if (Location > 700) {    temp = 700; flag = 1;}
   return temp;
}


//   ENSURE A  UNIT DOES NOT EXCEED Y LIMITS OF GRID MAP

double UnitType::CheckBoundsY(int &flag, const double Location)    {
   double temp = Location;
   if (Location <0) {    temp = 0;    flag = 1;}
   if (Location > 400) {    temp = 400; flag = 1;}
   return temp;
}


//   DETERMINE A COURSE TO REACH A GIVEN POINT

double UnitType::CalcCourse(double X, double Y){
   double Xval = (X - xLocation);
   if (Xval != 0) {
      return atan2(Y-yLocation,Xval);
   }
   else {
      return Y > yLocation?1.570796:-1.570796;
   }
}


double UnitType::Distance()    {
   double xcomp, ycomp;
   xcomp = xLocation - XP[WP];
   ycomp = yLocation - YP[WP];
   return sqrt((xcomp*xcomp)+(ycomp*ycomp));
}


//   DETERMINE COURSE AND SPEED TO REACH NEXT WAYPOINT AND THE TIME OF ARRIVAL

void UnitType::ChartCourse(const int T)    {
   Course = CalcCourse(XP[WP],YP[WP]);
   Speed = -1;
   while (Speed < 0) {
   Speed = (Type == 19)?MaxSpeed:RandNumb.Norm(MaxSpeed,1);
   }
   StopTime = T + floor(60*Distance()/Speed);
   Motion = 1;
}


//

void UnitType::NewSSMLoc() {
   int dummy = 0;
   XP[1] = CheckBoundsX(dummy,RandNumb.Norm(0,10) + xLocation);
   YP[1] = CheckBoundsY(dummy,RandNumb.Norm(0,10) + yLocation);
}
```

117

```
//   Edward R. Martinez
//   Thesis
//   September 1996
//   MSDOS 6.22
//   Borland C++ 4.02 for windows
//   THIS IS FILE Queue.h

//   This file models a Queue. It uses the class QueueElement to model an
//   element of the queue.  It is utilized by the class GridType in the file
//   Grid.cpp

#ifndef __queue_h
#define __queue_h

#include <iostream.h>
#include <fstream.h>
#include "Queuele.h"

//DEFINE CLASS queueType

class queueType {
    public :
        queueType() {nodeCount = 0; Head = Tail = NULL;}
        ~queueType() {Resetqueue();}
        void AddFront(const int, const int, const int, const double, const double);
        void Display(ostream&);
        int   queueEmpty() {return (nodeCount > 0 ? 0 : 1);}
        int   queueSize()   {return nodeCount;}
        int   RemoveMember(const int);
        void Resetqueue();
        int LookConf() {return LookPtr->GetConfidence();}
        int FindElement(const int);
        void SecondSight(const int, const double, const double);
        void InitLookPtr()    {LookPtr = Head;}
        void LookUpXY(double &,double &);
        int LookAtGrid(int&);
        int LookUnit() {return LookPtr->GetIdentifier();}
        int GetLookTime() {return LookPtr->GetDetectTime();}
        void IncLookPtr() {LookPtr = LookPtr->GetNext();}
        int NoLookAvail() {return LookPtr == NULL?1:0;}
    private :
        QueueElement *Head, *Tail, *LookPtr;
        int nodeCount;
    };

#endif
```

```
//   Edward R. Martinez
//   Thesis
//   September 1996
//   MSDOS 6.22
//   Borland C++ 4.02 for windows
//   THIS IS FILE queueType.cpp

#include <iostream.h>
#include <fstream.h>
#include "queue.h"
#include "Queuele.h"

void queueType::LookUpXY(double &Xp,double &Yp) {
    Xp = LookPtr->GetX();
    Yp = LookPtr->GetY();
}

void queueType::SecondSight(const int DTime, const double Xn, const double Yn)    {
    double Xadj, Yadj;
    Xadj = (LookPtr->GetX()*LookPtr->GetConfidence() + Xn)/(LookPtr->GetConfidence()+1);
    Yadj = (LookPtr->GetY()*LookPtr->GetConfidence() + Yn)/(LookPtr->GetConfidence()+1);
    LookPtr->BoostConfidence(DTime);
    LookPtr->SetXY(Xadj,Yadj);
}

int queueType::FindElement(const int Number) {
    InitLookPtr();
    for (int i = 0; i < queueSize(); i++)   {
        if (Number == LookPtr->GetIdentifier())    {return 1;}
        IncLookPtr();
    }
    return 0;
}

int queueType::LookAtGrid(int &Number) {
    int Temp;
    Temp = LookPtr->GetElement(Number);
    return Temp;
    }

//   Function Resetqueue

void queueType::Resetqueue()   {
    if(!queueEmpty()) {
        QueueElement *NEXTptr, *CurrentPtr = Head;
        while (CurrentPtr)    {
            NEXTptr = CurrentPtr->GetNext();
            delete CurrentPtr;
            CurrentPtr = NEXTptr;
        }
        nodeCount = 0;
        Head = Tail = NULL;
    }
}

//   Function AddFront

void queueType::AddFront(const int Newnode, const int NewValue, const int Timer,
                const double Xn, const double Yn)    {
        if (queueEmpty())
            Head = Tail = new QueueElement(Newnode,NewValue,Timer,Xn,Yn);
        else {QueueElement *temp;
                Head->SetPrevious(new QueueElement(Newnode,NewValue,Timer,Xn,Yn));
                temp = Head->GetPrevious();
                temp->SetNext(Head);
                Head = temp;
```
119

```cpp
            }
        nodeCount++;
    }


//   Function Display

void queueType::Display(ostream& method)   {

    if (!queueEmpty()) {
        QueueElement *CurrentNode = Head;
        do {
            CurrentNode->Display(method);
            CurrentNode = CurrentNode->GetNext();
        } while (CurrentNode);
    }
}


//   Function RemoveMember

int queueType::RemoveMember(const int ElementToFind)   {
    if (queueEmpty()) return 0;
    QueueElement *CurrentNode = Head;
    do {
        if (ElementToFind == CurrentNode->GetIdentifier()) {
            if (nodeCount == 1)   {
                delete CurrentNode;
                Head = Tail = NULL;
                nodeCount = 0;
                return 1;}

            if (CurrentNode == Head)    {
                Head = Head->GetNext();
                Head->SetPrevious(NULL);
                delete CurrentNode;
                nodeCount--;
                return 1;}

            if (CurrentNode == Tail)     {
                Tail = Tail->GetPrevious();
                Tail->SetNext(NULL);
                delete CurrentNode;
                nodeCount--;
                return 1;}

            QueueElement *TempBefore = CurrentNode->GetPrevious();
            QueueElement *TempAfter = CurrentNode->GetNext();
            TempBefore->SetNext(TempAfter);
            TempAfter->SetPrevious(TempBefore);
            delete CurrentNode;
            nodeCount--;
            return 1;
        }
        CurrentNode = CurrentNode->GetNext();
    } while (CurrentNode);
    return 0;
}
```

```cpp
//   Edward R. Martinez
//   Thesis
//   September 1996
//   MSDOS 6.2
//   Borland C++ 4.02 for windows
//   THIS IS FILE QueueElem.cpp

//   This file models an element of a Queue.  It is utilized by the class
//   dequeType in the file queuetype.cpp

#ifndef __QueueEle_h
#define __QueueEle_h

#include <iostream.h>
#include <fstream.h>

//DEFINE CLASS QueueElement

class QueueElement {
    public :
        QueueElement(const int Newint1, const int Newint2,const int DetTime,
        const double Xnew, const double Ynew)
            {
            int1 = Newint1;    int2 = Newint2;
            int3 = 1;          int4 = DetTime;
            Real1 = Xnew;      Real2 = Ynew;
            Real3 = 0.0;       Real4 = 0.0;
            NEXT = PREVIOUS = NULL;  }

        ~QueueElement() {NEXT = PREVIOUS = NULL;}
        void Display(ostream& method) {method<<int1<<"   ";}
        void SetNext(QueueElement *nextptr) {NEXT = nextptr;}
        void SetPrevious(QueueElement *previousptr) {PREVIOUS = previousptr;}
        int GetElement(int &EleValue) {EleValue = int2;return int1;}
        QueueElement* GetNext() {return NEXT;}
        QueueElement* GetPrevious() {return PREVIOUS;}
        void BoostConfidence(const int DetTime)   {int3++; int4 = DetTime;}
        void SetXY(const double Xnew, const double Ynew)    {Real1 = Xnew;Real2=Ynew;}
        int GetDetectTime()   {return int4;}
        int GetIdentifier()   {return int1;}
        int GetConfidence()   {return int3;}
        double GetX()   {return Real1;}
        double GetY()   {return Real2;}

    private :
        QueueElement *NEXT, *PREVIOUS;
        int int1, int2, int3, int4;
        double Real1,Real2,Real3,Real4;
    };

#endif
```

```cpp
//   Edward R. Martinez
//   Thesis
//   September 1996
//   MSDOS 6.22
//   Borland C++ 4.02 for windows
//   THIS IS FILE stack.h

//   This file models a stack. It uses the class stackElement to model an
//   element of the stack.  It is utilized by the class GridType in the file
//   Grid.cpp

#ifndef __stack_h
#define __stack_h

#include <iostream.h>
#include <fstream.h>
#include "stackele.h"

//DEFINE CLASS stackType

class stackType {
    public :
        stackType() {nodeCount = 0; Head = NULL;}
        ~stackType() {Resetstack();}
        void PUSH(const int, const int, const int, const int, const int,
                const double , const double );
        void Display(ostream&);
        int   stackEmpty() {return (nodeCount > 0 ? 0 : 1);}
        int   stackSize()  {return nodeCount;}
        int   POP(int&, int&, int&, int&, int&,double &,double &);
        void Resetstack();

    private :
        stackElement *Head, *Tail;
        int nodeCount;
    };

#endif
```

122

```
//  Edward R. Martinez
//  Thesis
//  September 1996
//  MSDOS 6.22
//  Borland C++ 4.02 for windows
//  THIS IS FILE stackType.cpp

//  This class models a stack. It uses the class stackElement.  It is
//  utilized by the class GridType in file Grid.cpp

#include <iostream.h>
#include <fstream.h>
#include "stack.h"
#include "stackele.h"


//  Function Resetstack

void stackType::Resetstack()   {
    if(!stackEmpty()) {
       stackElement *NEXTptr, *CurrentPtr = Head;
       while (CurrentPtr)    {
          NEXTptr = CurrentPtr->GetNext();
          delete CurrentPtr;
          CurrentPtr = NEXTptr;
       }
       nodeCount = 0;
       Head = NULL;
    }
}


//  Function PUSH

void stackType::PUSH(const int New1, const int New2, const int New3,
    const int New4, const int New5,const double New6, const double New7) {
       if (stackEmpty())
          Head = Tail = new stackElement(New1,New2,New3,New4,New5,New6,New7);
       else {
          Tail->SetNext(new stackElement(New1,New2,New3,New4,New5,New6,New7));
          Tail = Tail->GetNext();
       }
       nodeCount++;
}


//  Function Display

void stackType::Display(ostream& method)   {

    if (!stackEmpty()) {
       stackElement *CurrentNode = Head;
       do {
          CurrentNode->Display(method);
          CurrentNode = CurrentNode->GetNext();
       } while (CurrentNode);
    }
}


//  Function POP

int stackType::POP(int &Elem1, int &Elem2, int &Elem3,int &Elem4, int &Elem5,
    double &Elem6, double &Elem7)    {
    if (stackEmpty()) return 0;

    Elem1 = Head->GetElement1();
    Elem2 = Head->GetElement2();
    Elem3 = Head->GetElement3();
    Elem4 = Head->GetElement4();
```

```
    Elem5 = Head->GetElement5();
    Elem6 = Head->GetElement6();
    Elem7 = Head->GetElement7();


    if (nodeCount == 1)   {
        delete  Head;
        Head = Tail = NULL;
        nodeCount = 0;
        return  1;}

    stackElement *temp;
    temp = Head;
    Head = Head->GetNext();
    delete  temp;
    nodeCount--;
    return  1;

}
```

```
//  Edward R. Martinez
//  Thesis
//  September 1996
//  MSDOS 6.2
//  Borland C++ 4.02 for windows
//  THIS IS FILE StackEle.cpp

//  This file models an element of a Stack.  It is utilized by the class
//  stackType in the file stack.cpp

#ifndef __stackEle_h
#define __stackEle_h

#include <iostream.h>
#include <fstream.h>

//DEFINE CLASS stackElement

class stackElement {
    public :
        stackElement(const int Ele1, const int Ele2, const int Ele3,
           const int Ele4, const int Ele5, const double Ele6, const double Ele7) {
              Element1 = Ele1;Element2 = Ele2;Element3 = Ele3;Element4 = Ele4;
              Element5 = Ele5;Element6 = Ele6;Element7 = Ele7;NEXT = NULL;}

        ~stackElement() {NEXT =  NULL;}
        void Display(ostream& method) {
           method<<Element1<<"  "<<Element2<<" "<<Element3<<"  "<<Element4
                <<" "<<Element5<<"  "<<Element6<<" "<<Element7<<endl;}
        void SetNext(stackElement *nextptr) {NEXT = nextptr;}
        int GetElement1() {return  Element1;}
        int GetElement2() {return  Element2;}
        int GetElement3() {return  Element3;}
        int GetElement4() {return  Element4;}
        int GetElement5() {return  Element5;}
        double GetElement6() {return  Element6;}
        double GetElement7() {return  Element7;}
        stackElement* GetNext() {return  NEXT;}

    private :
        stackElement *NEXT;
        int Element1, Element2, Element3, Element4, Element5;
        double Element6, Element7;
};

#endif
```

```
//   Edward R. Martinez
//   September 1996
//   THESIS
//   MSDOS 6.22
//   Borland C++ 4.02 for windows
//   THIS IS FILE RdNumGen.h

//   This class was written by Arnie Buss, Naval Postgraduate School, and this
//   code is a specific section of his Random Number Generation Class


#ifndef __RdNumGen_h
#define __RdNumGen_h

#include <math.h>
#include <stdlib.h>
#include <time.h>

const long MODLUS = 21474836471;
const long MULT1 = 241121;
const long MULT2 = 261431;

class RdGen {
    public :
        RdGen();
        RdGen(const long);
        double Uniform();
        double Unif(const double , const double );
        long UnifI(const long a, const long b);
        double Normal();
        double Norm(const double , const double );

    private :
        long    StartingSeed,CurrentSeed;
        int     BoxFlop;
        double   BoxSave;


        };


#endif
```

```
//  Edward R. Martinez
//  September 1996
//  THESIS
//  MSDOS 6.22
//  Borland C++ 4.02 for windows
//  THIS IS FILE RdNumGen.h

//  This class was written by Arnie Buss, Naval Postgraduate School, and this
//  code is a specific section of his Random Number Generation Class.

//  The one major modification to his code is to allow the random number
//  generator seed to be the present computer clock time

#include <math.h>
#include <stdlib.h>
#include <time.h>
#include "RdNumGen.h"

RdGen::RdGen() {
    CurrentSeed = time(NULL);
    StartingSeed = CurrentSeed;
    BoxFlop = 1;
}

RdGen::RdGen(const long initSeed)    {
    StartingSeed = initSeed;
    CurrentSeed =  initSeed;
    BoxFlop = 1;
}

double RdGen::Uniform() {
    long zi, lowprd, hi31;
    zi     = CurrentSeed;
    lowprd = (zi & 655351) * MULT1;
    hi31   = (zi >> 16) * MULT1 + (lowprd >> 16);
    zi     = ((lowprd & 655351) - MODLUS) + ((hi31 & 32767) << 16) + (hi31 >> 15);

    if (zi < 0) {  zi += MODLUS;}
    lowprd = (zi & 655351) * MULT2;
    hi31   = (zi >> 16) * MULT2 + (lowprd >> 16);
    zi     = ((lowprd & 655351) - MODLUS) + ((hi31 & 32767) << 16) + (hi31 >> 15);

    if (zi <0) { zi += MODLUS;}

    CurrentSeed = zi;
    return double ((((zi >> 7) | 1) + 1)/16777216.0;
}

double RdGen:: Unif(const double a, const double b) {
    if ( a <= b) {return a + (b - a) * Uniform();}
    else { return (a + b)/2.0;}
}

long RdGen::UnifI(const long a, const long b)    {
    if ( a <= b)  {return   a + (long) floor((b - a + 1) * Uniform());}
    else  {return  0;}
}

double RdGen::Normal()   {
    double   v1, v2, w, y;

    if (BoxFlop)   {
        w = 2.0;
        while (w  > 1.0 || w < .0001) {
            v1 = 2.0 * Uniform() - 1.0;
            v2 = 2.0 * Uniform() - 1.0;
```
127

```
            w = v1 * v1 + v2 * v2;
        }
        y = sqrt( - 2.0* log(w) / w );
        BoxSave = v2 * y;
        BoxFlop = 0;
        return v1 * y;
    }
    else  {
        BoxFlop = 1;
        return BoxSave;
    }
}


double RdGen::Norm(const double mean, const double std)  {
    if (std <= 0.0) {return mean;}
    else {return mean + std * Normal();}
}
```
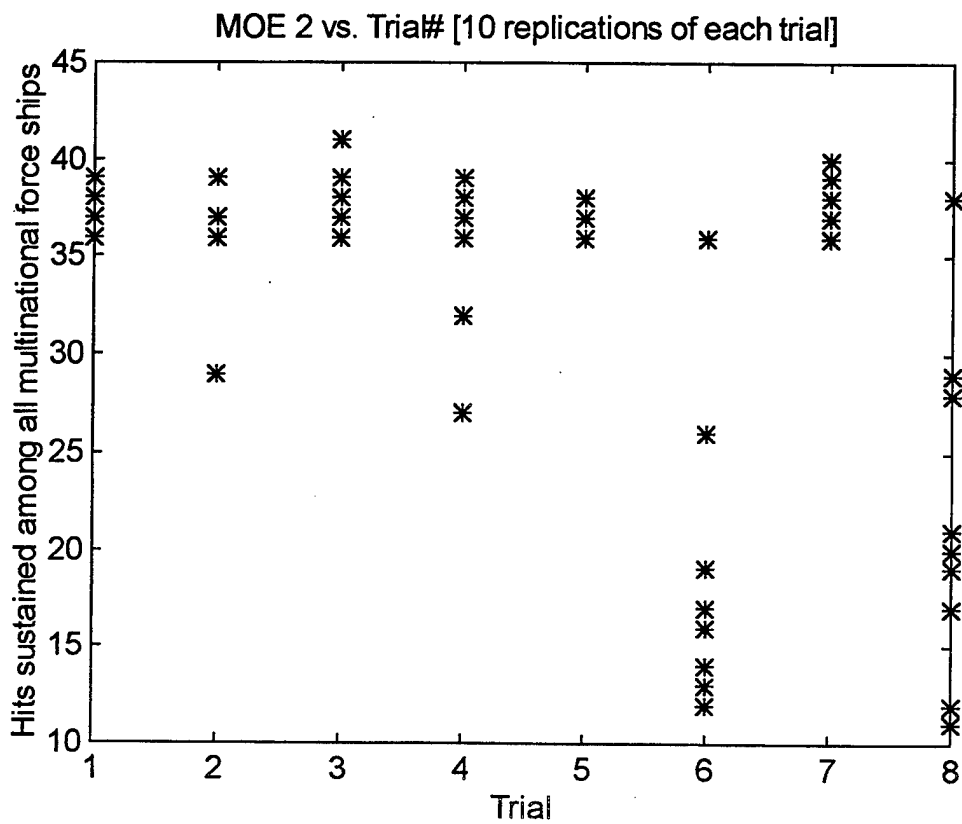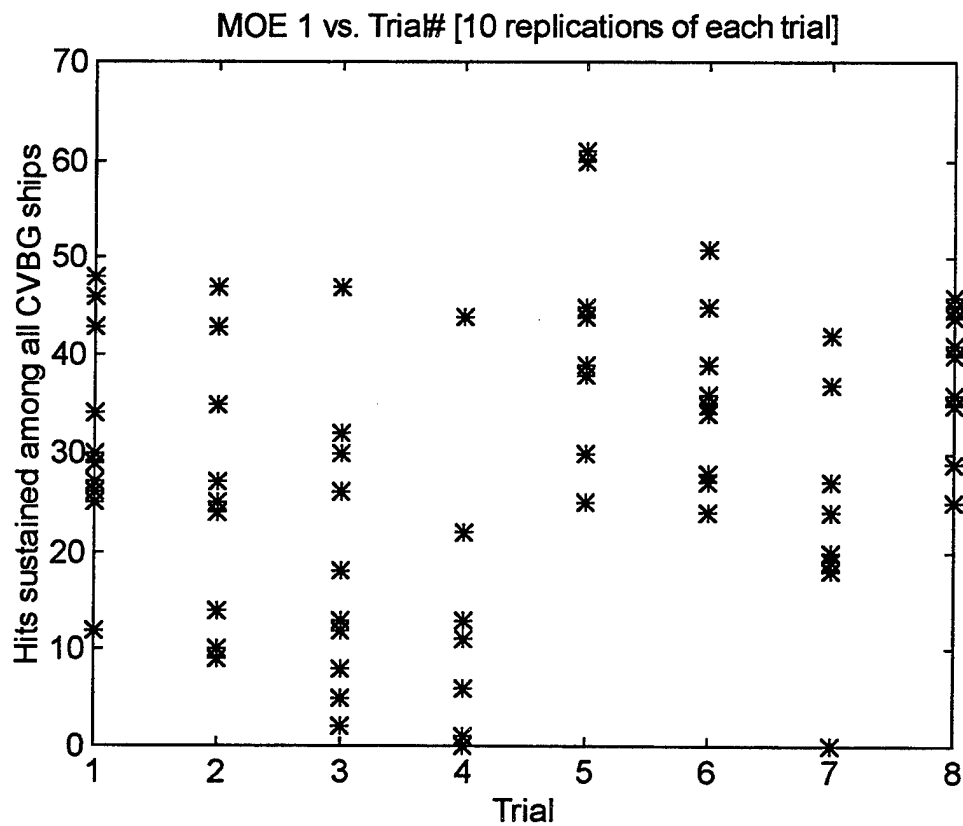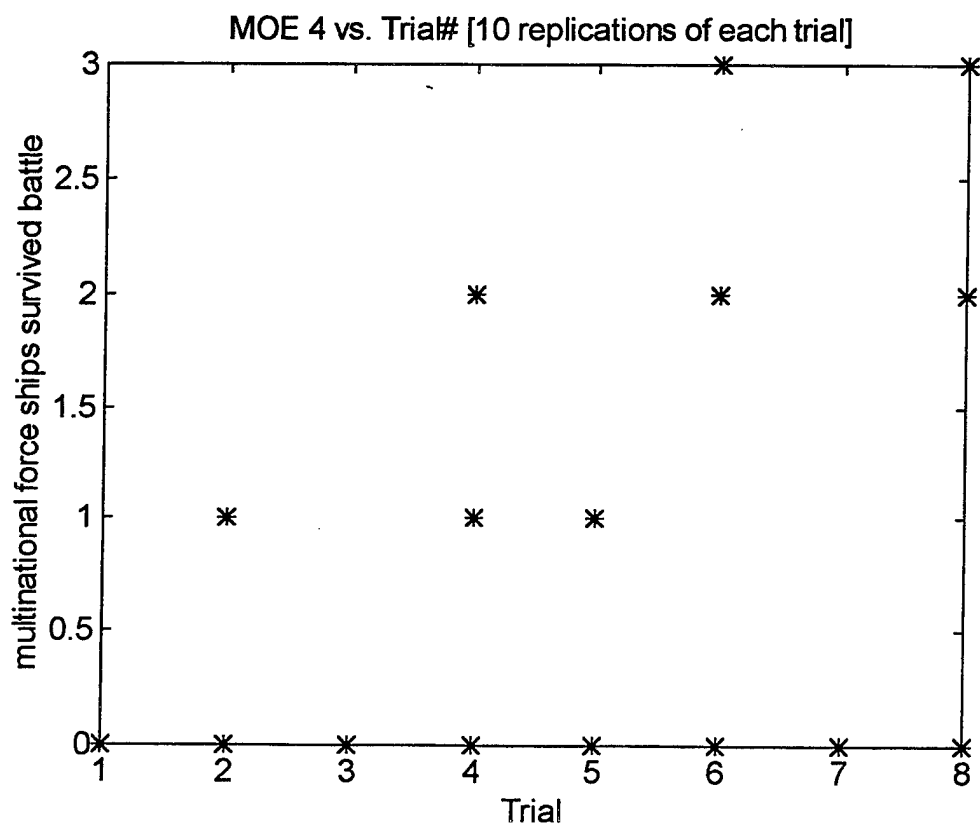
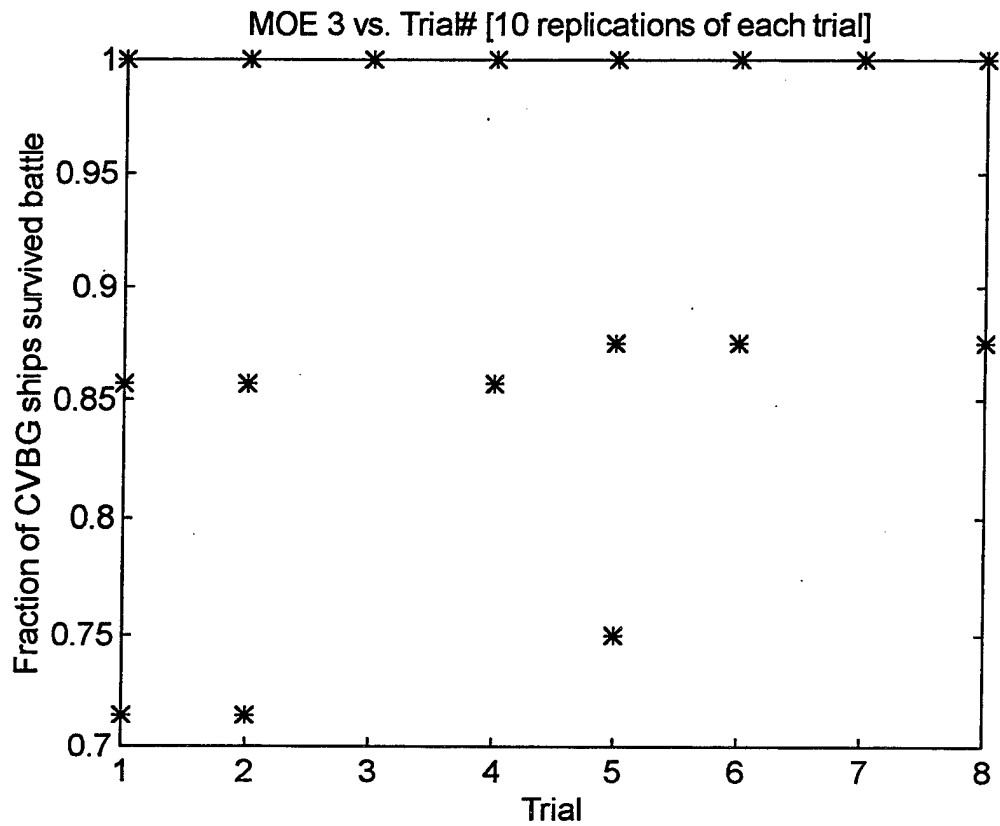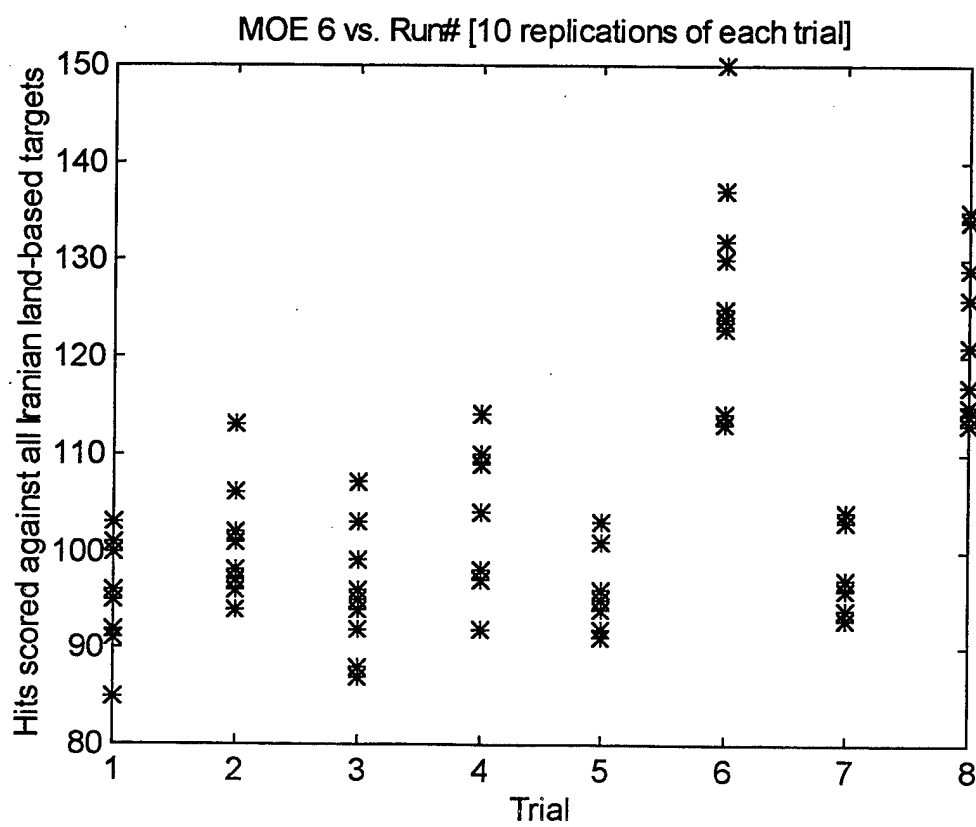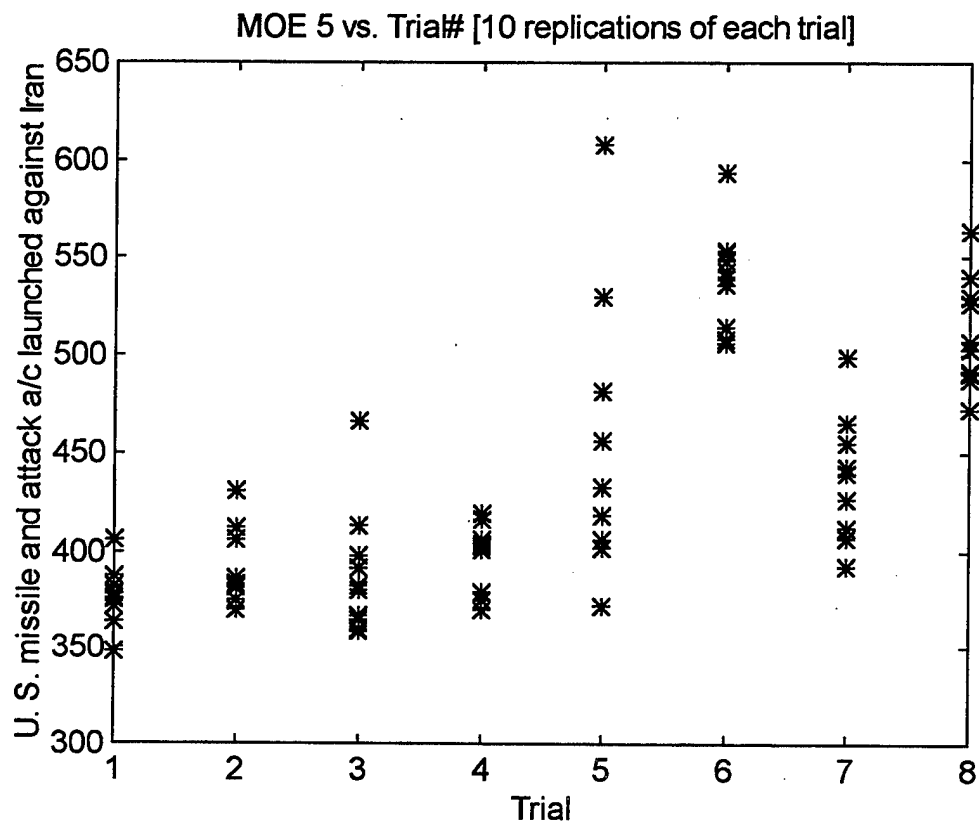## APPENDIX F. SIMULATION RUN DATA

trial	replication	moe 1	moe 2	moe 3	moe 4	moe 5	moe 6	moe 7
1	1	26	38	1	0	387	95	200
	2	46	37	1	0	378	101	176
	3	34	37	1	0	382	101	180
	4	48	37	0.857143	0	382	96	182
	5	12	37	1	0	375	96	134
	6	25	39	1	0	406	103	142
	7	27	39	1	0	364	100	176
	8	29	37	0.857143	0	372	92	186
	9	43	36	0.714286	0	348	85	198
	10	30	37	1	0	376	91	208
2	1	25	36	0.857143	0	375	94	132
	2	14	39	0.857143	0	381	98	132
	3	47	37	0.714286	0	383	96	194
	4	24	39	1	0	386	97	138
	5	35	39	0.714286	0	406	102	150
	6	43	37	1	0	370	106	180
	7	10	37	1	0	406	102	120
	8	9	29	1	1	430	113	106
	9	27	39	0.857143	0	382	96	162
	10	35	36	0.857143	0	412	101	168
3	1	13	39	1	0	391	96	164
	2	5	38	1	0	360	88	134
	3	24	41	1	0	367	92	150
	4	47	36	1	0	380	107	188
	5	26	37	1	0	398	94	156
	6	8	36	1	0	466	99	146
	7	2	38	1	0	413	103	120
	8	18	36	1	0	382	103	146
	9	12	38	1	0	365	95	138
	10	32	38	1	0	358	87	150
4	1	0	32	1	1	416	110	114
	2	13	36	1	1	375	98	136
	3	13	37	1	0	401	104	140
	4	1	27	1	2	419	114	84
	5	1	36	1	1	404	110	106
	6	6	38	1	0	419	110	118
	7	0	32	1	1	379	97	102
	8	22	39	1	0	403	110	148
	9	44	39	0.857143	0	370	92	190
	10	11	39	1	0	406	109	128
5	1	61	36	1	0	608	96	206
	2	25	38	1	1	418	96	156
	3	44	36	1	0	482	95	216
	4	60	38	0.75	0	372	91	208
	5	38	37	1	0	456	101	194
	6	39	38	1	0	530	103	194
	7	45	38	0.875	0	406	94	176
	8	25	37	1	0	432	91	166
	9	30	36	1	0	406	92	178

Trial	replication	moe 1	moe 2	moe 3	moe 4	moe 5	moe 6	moe 7
5	10	25	36	1	0	402	91	162
6	1	39	16	1	3	543	137	132
	2	34	12	1	3	548	125	104
	3	36	19	1	3	515	123	122
	4	45	26	1	2	508	124	158
	5	27	26	1	2	540	123	122
	6	24	13	1	3	536	130	96
	7	51	36	0.875	0	506	115	162
	8	39	14	1	3	594	113	128
	9	35	14	1	3	553	114	134
	10	28	17	1	3	554	132	98
7	1	27	37	1	0	465	96	168
	2	0	38	1	0	412	97	140
	3	37	37	1	0	455	104	174
	4	20	37	1	0	499	103	180
	5	24	37	1	0	443	94	152
	6	27	37	1	0	407	94	158
	7	19	38	1	0	443	93	140
	8	27	39	1	0	440	97	172
	9	42	40	1	0	426	103	192
	10	18	36	1	0	392	94	114
8	1	41	11	0.875	3	540	134	124
	2	36	17	1	3	507	117	112
	3	25	29	1	2	491	121	118
	4	44	21	1	3	503	126	140
	5	46	20	0.875	3	488	113	128
	6	40	28	1	3	530	129	162
	7	45	19	0.875	3	472	114	138
	8	29	38	1	0	492	115	142
	9	44	17	1	3	564	135	104
	10	35	12	1	3	527	135	112

MOE 1 vs. Trial# [10 replications of each trial]

MOE 2 vs. Trial# [10 replications of each trial]

MOE 3 vs. Trial# [10 replications of each trial]

MOE 4 vs. Trial# [10 replications of each trial]

MOE 5 vs. Trial# [10 replications of each trial]

MOE 6 vs. Run# [10 replications of each trial]

MOE 7 vs. Trial# [10 replications of each trial]
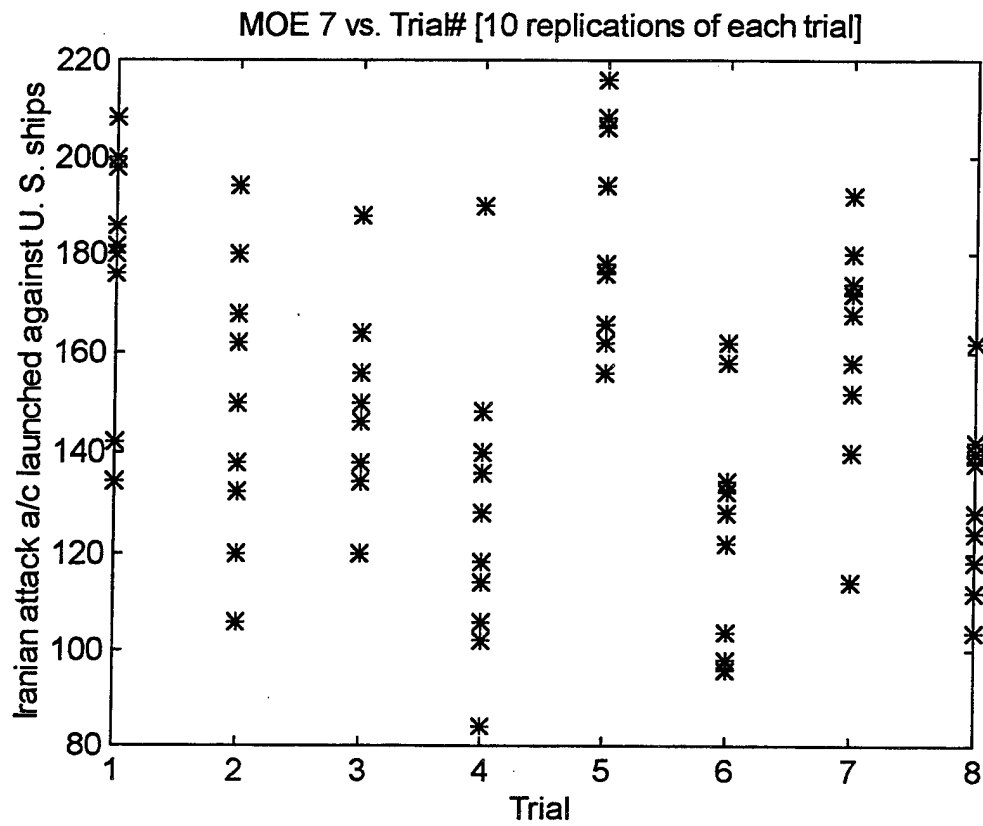
INITIAL DISTRIBUTION LIST

No. of copies

Defense Technical Information Center 2
8725 John J. Kingman Road, Ste 0944
Ft. Belvoir, VA 22060-6218

Dudley Knox Library 2
Naval Postgraduate School
411 Dyer Road
Monterey, CA 93943-5101

Professor Donald Gaver, Code OR/Gv 1
Department of Operations Research
Naval Postgraduate School
Monterey, CA 93943-5101

Professor Patricia A. Jacobs 1
Department of Operations Research, Code OR/Jc
Naval Postgraduate School
Monterey, CA 93943-5101

Lieutenant Edward R. Martinez 2
17912 Beach Street
Umatilla, FL 93940